Technical Report 1735
January 1997

Automated Integrated Communications Systems (AICS) Integrated Network Manager Prototype Documentation

E. W. Jacobs M. E. Inchiosa L. M. Gutman C. T. Barber

19970210 147

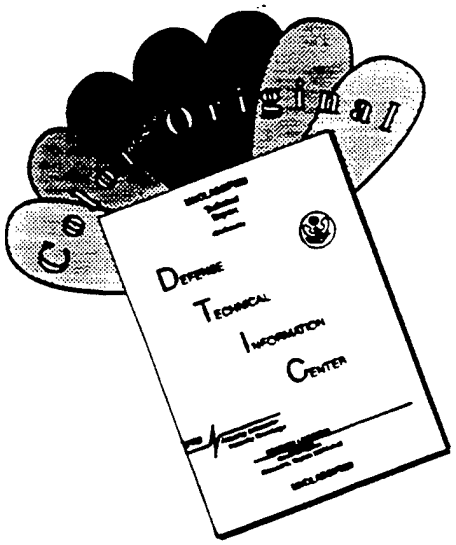
Naval Command, Control and
Ocean Surveillance Center
RDT&E Division

San Diego, CA
92152–5001

DTIC QUALITY INSPECTED 1

Approved for public release; distribution is unlimited.

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

Technical Report 1735
January 1997

Automated Integrated Communications Systems (AICS) Integrated Network Manager Prototype Documentation

E. W. Jacobs
M. E. Inchiosa
L. M. Gutman
C. T. Barber

NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152-5001

H. A. WILLIAMS, CAPT, USN
Commanding Officer

DR. R. C. KOLB
Executive Director

ADMINISTRATIVE INFORMATION

The work detailed in this report was performed for the Networking Design and Analysis Branch (Code D827) of the Communications Systems Engineering and Integration Division (Code D82) of the Communications Department (Code D80) of the Research, Development, Test and Evaluation Division of the Naval Command, Control and Ocean Surveillance Center by the Ocean Survey Systems Branch (Code D364). Funding was provided by the Space and Naval Warfare Systems Command (SPAWAR PD 13, LCDR Glenn Darling) under program element 0603794N. This report covers work performed from December 1995 to September 1996.

Released by
R. L. Merk, Head
Networking Design and Analysis Branch

Under authority of
R. J. Kochanski
Communications Systems
Engineering and Integration
Division

ACKNOWLEDGMENTS

The authors wish to thank Brian Clingerman of SPAWAR 176E for his support and interest in this project.

SB

EXECUTIVE SUMMARY

This report documents the Integrated Network Manager (INM) prototype development and integration effort, undertaken as part of the FY96 Automated Integrated Communications Systems (AICS) program. The AICS architecture describes a hierarchy of INMs, where local operations centers (LOCs), the INMs at the at the lowest level of the hierarchy, are responsible for the communications/network assets under their purview. The AICS architecture is primarily targeted for an environment where INMs are commonly separated from other INMs by RF links. The INM prototype work centered on development of the AICS Management Application (AMA). The AMA is a network management application that provides several Navy-specific functions to the INM. These functions include the following:

- Managing the network within the framework of the AICS architecture;
- Managing the network based on mission specific communication plans and policy rules;
- Providing a standards-based method to describe aggregate network status information;
- Presenting Navy operators with an understandable description of the network status in the context of the mission communications plan;
- Facilitating integration of COTS network management programs into the INM.

The ultimate vision for AICS is a highly automated network management system where its functions are implemented using standards-based management protocols, and where many operations are carried out in a virtually unattended mode. The prototype INM provides a working example demonstrating aspects of this ultimate vision, and provides a starting point for development of a network management system that can evolve with new technology.

CONTENTS

FIGURES

1. INTRODUCTION

As part the FY96 AICS program, a prototype Integrated Network Manager (INM) has been developed. The prototype INM represents both a development and integration effort. The requirements of the INM are described in the AICS *Network Management Architecture* (NMA) and *Network Management System Segment Specification* (SSS) documents (reference 1). The AICS Management Application (AMA), a component of the prototype INM, is an application developed to provide requirements delineated in the NMA and SSS which are specific to Navy networks, requirements that are not currently provided by COTS products. The AMA was designed to facilitate integration of COTS products so as to minimize the development of custom software, and to take advantage of COTS products when they do fulfill INM requirements. To allow for flexibility in implementation, the INM prototype design is modular, with the AMA designed such that it is not restricted to integration with specific COTS products. The purpose of the INM prototype is to provide the developers of a fieldable AICS implementation some clarification on how to interpret the requirements contained in the NMA and SSS, and to provide a good starting point for future implementation efforts.

The NMA and SSS describe an architecture where an AICS *Integrated Network Manager* (INM) at the bottom of a chain of INMs is responsible for the communications/network assets under its purview (i.e., that are part of its *unit*). The bottom level INM is referred to as a Local Operations Center (LOC). Units that contain higher level INMs (e.g., Regional Operation Centers (ROCs) or Network Operation Centers (NOCs)) will also contain an LOC which will directly manage the local units communications/network assets. The AICS architecture is primarily targeted for an environment where INMs are commonly separated from other INMs by RF links.

The majority of INM tasks occur at the LOC level of the management hierarchy, while at the ROC and NOC level, a main INM function is to present a display detailing subordinate status information. As a result, much of the work during FY96 has centered on the INM LOC prototype. A brief summary of a LOC's required functionality is as follows:

1. Receive *communication plans* and *policy rules statements* from higher level INMs;
2. Configure local assets in accordance with the communications plan and policy rules;
3. Monitor status of local assets in accordance with the communication plan and policy rules, and provide status information relevant to the current mission in a format understandable to the present operator;

4. Report filtered management information to higher level INMs. In accordance with policy rules, reporting will be done on a periodic basis, a fault-driven basis, and/or upon request.

The *communication plans* and *policy rules statements* introduced in item 1 are central to the design of the AMA. The communications plan describes the equipment, services, resources, applications, and management hierarchy for the current mission. As a basis for the AMA, a Simple Network Management Protocol (SNMP) Management Information Base (MIB) has been written which encompasses the information contained in a communications plan. An SNMP agent associated with the AMA called the AMA Agent (AA) implements this MIB. Included in the MIB are objects describing the communications plan and the operational status of all the elements of the communications plan. The MIB is written hierarchically, the result being that the operational status objects represent aggregate status ranging from the status of very specific network elements to the overall operational status of the mission. At the start of a new mission, the values of a set of MIB objects are set in accordance with the communications plan for the new mission, and the AA is initialized. At initialization, the other AMA inputs are the policy rules. The policy rules describe how the elements of the communications plan are to be managed. The interplay of the communications plan and the policy rules will be described in more detail in subsequent sections.

As indicated in item 2, the NMA and SSS require that the INM perform configuration management tasks. Because of the lack of a standards-based authentication mechanism, typical COTS SNMP agent software does not permit configuration. This problem will be solved when authentication is incorporated into SNMP (which will occur relatively soon), and when the support for authenticated SNMP becomes commonly available in COTS products (which could take considerably longer). In the INM prototype, the AMA performs some automated configuration management tasks, although these tasks are limited to configuration of network elements closely associated with the INM itself. More details on configuration of the COTS Mid Level Manager (MLM) and on the Remote Monitoring (RMON) probe are given in subsequent sections. It is envisioned that, as standards-based configuration of COTS devices becomes implementable, automated configuration of network devices and applications in accordance to the communications plan and policy rules will become a more important function of the INM.

Items 3 and 4, monitoring of the network and reporting of required information based on the communications plan and policy rules, are demonstrated in the INM prototype. Subsequent sections will detail how the communications plan, policy rules, integration of the AMA with COTS products, and SNMP are utilized to accomplish these tasks.

The ultimate vision for AICS is a highly automated system where functions 1 through 4 (along with other required functions as described in the NMA and SSS

documents) are implemented using standards-based management protocols, and may be carried out in a virtually unattended mode. The prototype INM documented here is intended to give a preview of this ultimate vision, and to lead AICS network management development toward a flexible design that can evolve with the available technology.

Section 2 of this document provides details of the INM prototype, and section 3 provides a summary and concluding remarks. Two appendices are also included. Appendix A is the AMA Agent MIB that is discussed in section 2.2.1, and Appendix B provides some selected code that is discussed in section 2.2.2.

2. INM PROTOTYPE

2.1 Overview

Figure 1 helps explain the design of the prototype INM by means of an operational example. The figure shows a sequence of events indicated by the numbered bubbles. In step 1, a higher level INM (e.g., a ROC), as a result of operator action (using the AMA interface) or possibly as an automated response from yet a higher level INM, sends a message to the the local INM LOC indicating it should retrieve the appropriate information from its databases to initiate the communications plan for a new mission. The LOC receives the messages, checks that the message is authentic (i.e., that the message actually came from a superior ROC, and that the message received is identical to the message that was sent), displays a message to alert the local operator, and loads the communications plan by initializing the AA. Configuration associated with the communication plan then begins.

In the current prototype, four different configuration processes are performed. Each one of these configuration processes is performed as the communications plan and the policy rules require. The elements of the communications plan relevant to each particular configuration process are identified, and the configuration is performed in accordance with the applicable policy rules.

The first configuration performed is to create handles to enable SNMP communication between AMAs and other network entities that communicate with the AMA. This includes SNMPv2-USEC (references 2 and 3) handles to provide authenticated communication between AMAs, SNMPv1 handles so as to provide information to SNMPv1 management entities, and a trap daemon to enable the AMA to receive traps from SNMPv1 agents. As part of this configuration, processes for periodic reporting (SNMPv2 polling) and fault driven reporting (SNMPv2 informs) between the members of the INM hierarchy are set up and initiated.

The second configuration performed is to establish the dependency of the operational status of elements of the communications plan (which are represented by operational status mib objects). The indexing of objects in the mib is such that a hierarchy of operational status objects exists. The dependence between the operational status objects is set up based on the communications plan and a set of policy rules.

The third configuration performed is to configure periodic polling of network elements, filtering of the resulting responses from network elements, and filtering of unrequested messages (i.e., SNMP traps) from network elements. Once again, the communications plan is consulted to ascertain what network elements are utilized in the current mission, and a set of policy rules are consulted to determine the polling and the response and trap filtering procedure to employ. There are many ways one could implement this polling and filtering process. For instance, building this capability directly into the AMA would be a straightforward task. Alternatively, a variety

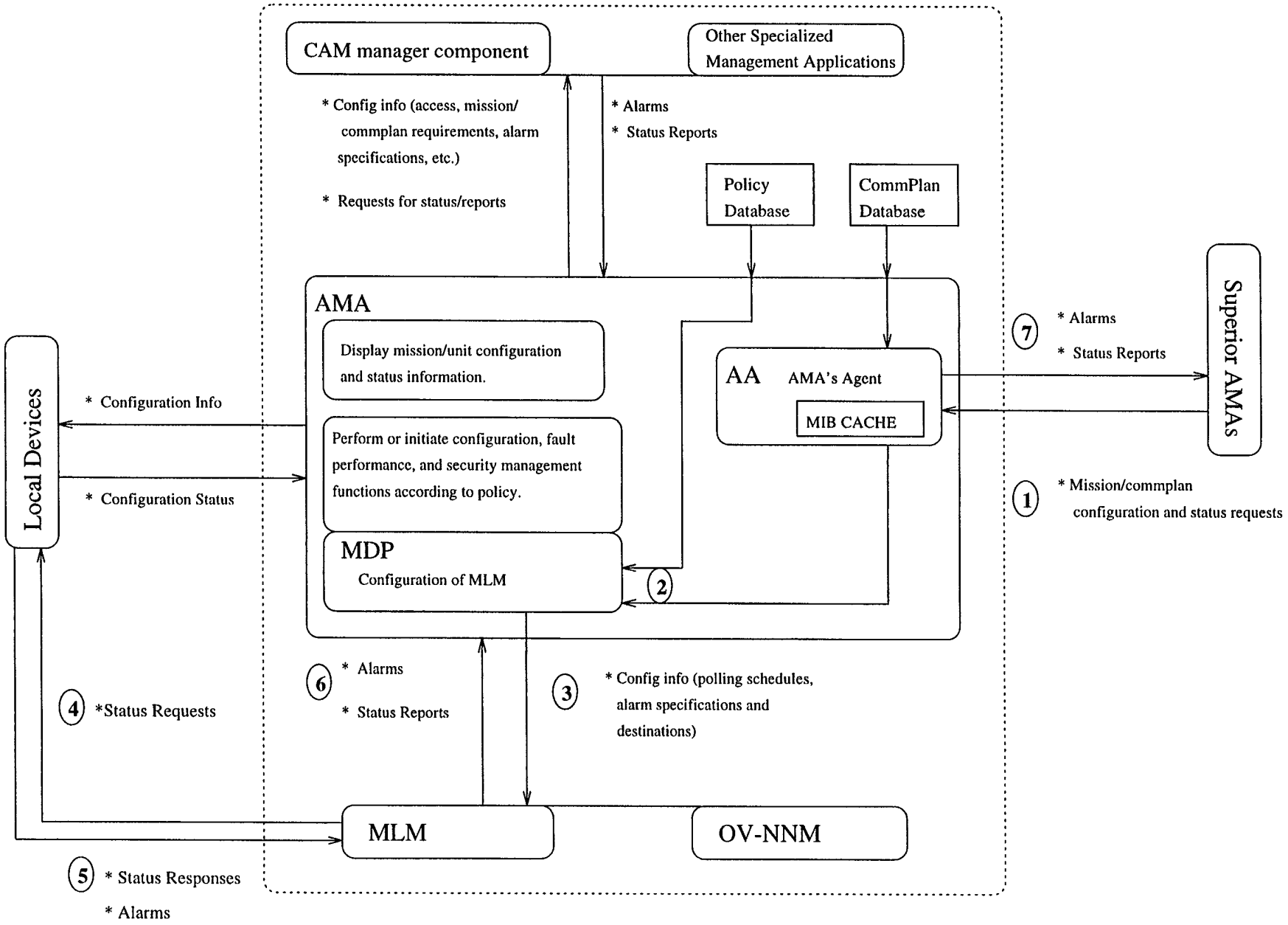


Figure 1. An INM operational example. See the text for a description of the numbered sequence.

of COTS products of varying capabilities could be employed. In this case it would be the AMA's task to configure and start the COTS program. As indicated by steps 2 and 3 in figure 1, in the current prototype a COTS mid-level manager (MLM) is used to fulfill the polling and filtering requirement. A module of the AMA program, labeled the management doctrine procedures (MDP) in figure 1, extracts elements from the communications plan where monitoring will be required, applies the policy rules relevant to the monitoring of these elements, and (re)configures and (re)starts the MLM.

Once the AMA is configured, the operator is presented with a display showing the hierarchy of INMs, and the operational status of the network elements relevant to the current communications plan. This display presents the underlying status of the network as it relates to the communications plan in terms that are understandable to the Navy operator. This display is not meant to replace the display provided by the SNMP Management platform program. The AMA's display does not provide a network map, auto-discovery, or other features that are already provided by such programs. Continuing with the example of figure 1, the MLM, now properly configured for the current mission, monitors the network by carrying out scheduled polling of network assets (step 4), and stands ready to process responses and traps it receives (step 5). When appropriate, the MLM then sends or forwards appropriate traps to the local AMA (step 6).

In response to received traps, the AMA then modifies the value of appropriate operational status objects in the MIB cache. The value of operational status objects which are dependent on the modified objects are recomputed. Based on the fault reporting rules configured for the current mission, SNMP inform requests are sent to superior INMs (step 7). The AMA and SNMP management platform program (HP OpenView Network Node Manager (reference 4) (OV-NNM) in figure 1) status displays are updated. In the current prototype, an object representing the AMA application has been incorporated into the OV-NNM topology display. The status of the symbol tracks the overall mission operational status as determined by the AMA. An operator who is observing the OV-NNM topology display would be alerted to status changes as perceived by the AMA. In addition, alarm messages may pop up on the LOC display to alert the operator to alarm conditions and suggest an appropriate course of action. Ultimately automated fault response systems may have to be integrated into the system as well.

This example briefly introduces the role of the AMA, AA, MLM, and the SNMP management platform program. It also introduces how the communication plan and policy rules are used to govern how INMs communicate, how operational status objects depend on one another, and how monitoring of local assets at the LOC level is performed. Not included in this example, but supported by the current prototype INM is a fourth configuration process, configuration of a Remote Monitoring (RMON) probe. In the following sections, a more detailed look at various components of the

INM is present, with concentration on the development, integration, and operation of the AMA.

2.2 AICS Management Application

COTS SNMP network manager platform programs such as HP OpenView Network Node Manager (OV-NNM) provide an SNMP management platform around which an INM can be built. Typically, more specialized network applications are layered on top of such a platform, often using the platforms application programming interface (API). The AMA is one such specialized network application. Because the AMA is a prototype effort, no commitment to dependence on a particular COTS SNMP management platform API was made. Instead, development of the AMA was done so that it might be loosely integrated with any SNMP management platform. The NMA and SSS require secure communication between INMs. If SNMP is to be used for INM to INM communications, a version of SNMPv2 supporting authentication would be required. As SNMPv2 with security features is still bogged down in the standardization process, there is not a wide variety of development tools available that support it. The desirability of loose coupling to a particular COTS SNMP management platform, and the requirement of authenticated INM communications were two compelling reasons to use the Tcl/Tk extension, *Scotty*, for in-house developed components of the prototype INM. Tcl/Tk is a powerful, highly portable, public domain scripting language that allows for customized extensions to be incorporated into the language (reference 5). *Scotty* is a Tcl/Tk extension for writing network applications (reference 6). *Scotty* provides the tools necessary to write both SNMP management and agent applications supporting both SNMPv1 and SNMPv2-USEC protocols.

A functional diagram of the AMA is provided by figure 1. The AMA is the focal point to which management information flows from higher level INMs, and from which management information flows to local devices and/or local device management applications. The AMA takes the communications plan and policy rules as input, and proceeds to carry out its functions as summarized in the previous section.

2.2.1 AA Management Information Base

The current version of the AA MIB is included in appendix A. The MITRE report *AN/SSQ-33(V)3 Control and Management Segment Information Reference Model (IRM)* (reference 7) was used as a starting point for the development of the AA MIB. The IRM describes a vision of what information might be needed to support the required functionality of the AICS management software. Furthermore, the IRM was developed to be used as the starting point for development of a MIB similar to that required for the AMA. Included in the IRM are management objects associated with the INM itself (e.g., objects related to management hierarchy and security), as well as objects describing the unit's network/communication assets. A syntactic conversion of the suggestions in the IRM to a prototype AA MIB would have been a relatively straightforward task, but would have introduced problems. The IRM in-

troduced some general concepts (e.g., "Network Domains") that could be used to characterize the various aspects of the network. To facilitate implementation, some of these general concepts were abandoned in favor of more focused definitions. As a result, some of the broad functionality inherent in the IRM was lost in favor of a MIB for which a clearer path to an implementation could be seen.

The AA MIB is quite different from the typical MIB written to support a specific hardware device. If one was to consider that the requirements outlined in the NMA and SSS represent a sort of application, then the AA MIB could be considered an application MIB. Again, the AA MIB is quite different from a typical application MIB in that the "application" is not rigorously defined. Therefore, the need or appropriateness for using the AA (and AA MIB) as a component of the AMA could be argued. Several compelling reasons were factors in choosing to use an SNMP Agent as the basis for the AMA. First, the AA provides a means for using a standards-based protocol, SNMP, for AMAs to use to communicate with each other. As SNMP is layered on User Datagram Protocol (UDP) and is specifically designed to conserve bandwidth, and as INMs are commonly separated by low-bandwidth RF links, using SNMP for this purpose is particularly appropriate. Second, the AA makes management information relevant to the AICS management tasks available using a standards-based protocol. Included is aggregate status information relevant to the current mission, information defined and maintained using standards-based methods. Third, the AA MIB may serve as a standards-based way of defining the requirements of the AMA (i.e., not only for the prototype, but for development of a fieldable AICS system). In theory, if future versions of the AA MIB become more robust, the AA MIB could serve to concisely define the AMA interface, enabling independent development of inter-operable AMAs. In light of this, the idea of employing an AA MIB and some of the concepts which have been incorporated into the version used in this prototype may be the most significant contributions of this work.

The AA MIB was designed to hold management information laid out in the context of a unit's current mission. Each table in the MIB contains an object whose value represents the aggregate operational status of the network element identified by the row in the table. The indexing of the MIB is hierarchical so that a dependency exists from the highest level aggregate operational status to the aggregate statuses of more detailed aspects of the network. In this way, the dependency of the overall operational status of the mission on underlying mission critical applications, the dependency of the mission critical applications on communications services, the dependency of the communications services on the lower level network assets, etc., is made available to the operator.

From a quick review of the AA MIB in appendix A, it is seen that the MIB contains a mission definition group (`missionDef`), mission communications plan group (`missionCommPlan`), unit definition group (`unitDef`), and unit network group (`unitNet`). The mission definition group defines objects that provide a high-level description of

the unit's current mission. Included in this group is the mission identification number (identifying which communications plan from the communications plan database is currently in use), the overall operational status of the current mission, and tables describing the INMs which are superior and subordinate to the local unit. The mission communication plan group contains two tables. The first table describes the user applications for the current mission. The second table describes the communications services these user applications require, and identifies which transmission resources they utilize. With respect to the OSI seven-layer network model (reference 8), the user application table is most closely associated with layer seven, and the communications services table is most closely associated with layers three and four.

The unit definition group defines objects that provide a high-level description of the local unit. Included in this group is the unit identification number, the type of the unit and the unit's INM, and a table describing the components of the unit's INM. The unit network group is a collection of tables that describes the communications and network assets under the INM's control. The highest level table in the unit network group is the transmission resource table. A transmission resource is, for instance, a particular RF network. The transmission resource is the aggregate of all the equipment, from the link and physical layers, that constitute the particular RF network. On a LOC, the transmission resource would consist of the aggregate of all the equipment on the local unit that constitutes the particular RF network. On a ROC, the transmission resource would consist of this aggregate over all the units subordinate to the ROC. Several other tables are included in the unit network group, each one indexed such that a hierarchy exists extending from rather detailed aspects of the equipment and configuration of the network, all the way to a very high-level description (e.g., a transmission resource). The tables in the unit network group are most closely related to layers one and two of the seven-layer OSI network model.

The indexing of the communication services table connects the higher layer tables in the mission communications plan group into the hierarchy existing in the unit network group. Each of the aforementioned tables contains an object for the value of the operational status of the network element identified with the row. These operational status objects are important to the operation of the AMA, in that they are aggregate status information that is of interest to the network operator. The last group of the AA MIB is the INM notification group (`inmNotification`). The INM notification group contains tables detailing the dependency of operational status objects, and detailing the rules for generation of alarms (i.e., inform requests) between INMs. The use of the INM notification group in conjunction with the operational status objects will be discussed in more detail in the following sections.

Examination of appendix A reveals that the `missionSec`, `missionConfig`, `unitSec`, and `unitOrg` groups have not been written. In addition, the current version of the prototype has not utilized the `unitLogPortTable`, `unitLinkTable`, or `unitChanTable`. A more complete understanding of the operation of the actual running networks for

which the AMA is targeted is required before work at this level of detail would be worthwhile.

2.2.2 AMA Design and Operation

Appendix B provides some selected AMA code intended to give the reader insight into the AMA prototype design and operation (reference 9). The code presented in appendix B is not intended to give a detailed understanding of the program's operation (even an experienced Tcl programmer would have difficulty doing this). The code is clear enough that only basic programming experience is required to obtain a general idea of its function. The main program for the AMA is in section B.1. A sample AMA configuration file is given in section B.2. Note that some important program parameters are set in this configuration file, including the following: whether or not the AMA should use authenticated communications between INMs; the name of the MIB files to read; and a list of the items the AMA should attempt to configure. In this sample file there are four entries in the configurable items list:

a. configuration of the AMA's INM to INM communication;

b. configuration of a mid-level manager;

c. dependence of operational status variables; and

d. configuration of an RMON probe.

When initiating the main program, the name of the communications plan to implement is passed as a parameter. Towards the middle of the program (section B.1), the AMA configuration file is read, and the MIB files are loaded. The name of the communications plan to implement is then passed to a procedure that initializes the AMA agent. Using the *Scotty* toolkit, the coding of the agent initialization procedure was relatively straightforward, although this part of the code is rather lengthy and hardly understandable to those not familiar with *Scotty*. Therefore, it is not included in Appendix B. In short, this procedure reads the communications plan, sets the values of the MIB objects which describe the communications plan, stores these values so that they may be accessed via SNMP requests, sets up access restrictions associated with SNMPv1 community strings and SNMPv2-USEC users and contexts, and sets up the bindings (i.e., the code to run) associated with writeable MIB objects.

Referring again to section B.1, note that after the agent is initialized, a procedure is called (`Config_Check`) that initiates configuration of the entries in the configurable items list. This part of the code is discussed below. After `Config_Check` has run, the final step of the main program is to start the AMA display. After the AMA display starts, the program responds to inputs from its SNMP interface and the AMA display.

The code for the procedure `Config_Check` is given in section B.3, and is quite simple. A stanza of code exists for each of the entries in the configurable items list.

In the simplest cases, this section of code simply calls the procedure `Configuration`. In the case of the mid-level manager, first a check is made to see if the local INM has a mid-level manager, and if so, the procedure `Configuration` is called (from a separate process started by `Config_Check`). The procedure `Configuration` will be discussed in the next section. If one wanted to extend the capabilities of the AMA (e.g., to initiate the configuration of other network elements), the only required changes to the code presented up to this point would be to add entries to the configurable items list in the AMA configuration file, and to add the corresponding stanza to the procedure `Config_Check`.

2.2.2.1 Management Doctrine Procedures

A fair amount has been written on implementing network management policies (references 10, 11, and 12). An attempt was made to draw upon this literature in hopes of arriving at a relatively intelligent approach to the AMA's implementation of policy. Application of management policies, or management doctrine, is an important concept incorporated into the AMA, therefore, in figure 1, the management doctrine procedures (MDP) are shown as a separate component of the AMA. The current prototype supports four configurable items that are processed by the MDP. The function of these configurable items are quite different, therefore, it was necessary that the MDP design allow for a great deal of flexibility. Furthermore, the MDP should be such that additional entries can be made in the configurable items list without having to change existing code. Therefore, the MDP was designed with extensibility as a primary concern.

The procedure `Configuration` presented in section B.4, is the top procedure of the MDP. As discussed in the previous section, it is called once for each of the entries in the configurable items list. Also of importance to the MDP are two configuration files holding policy information, specifically, the properties data base file and the policy rules file. Samples of these files for the INM SNMP session configuration are shown in sections B.5 and B.6, respectively.

To understand how the MDP works, the concept of *property groups* and *properties* must be introduced. The methodology of property groups and properties is used by the COTS network management program Nerve Center (reference 13). The methodology facilitates a flexible application of policy rules, and has been borrowed and adapted for the AMA. A property group is simply a list of properties. Network elements that possess the properties in a property group are said to be members of the property group. The properties data base file associated with INM SNMP session configuration (section B.5) shows several properties groups, with a list of properties associated with each property group (in this case there is only one property in each list). In the context of network management and the AMA specifically, management policy rules are associated with properties, and are applied to network elements that are members of a property group containing the given property. The policy rules file (section B.6)

shows a list of rules associated with INM SNMP session configuration. Each rule includes a policy template, and identifies the property that the network element must possess for the rule to be applicable. The policy template includes labels (which are associated with the type of network element that has the given property), and additional parameters that the MDP will need to correctly implement the policy. The syntax of the policy rules and the policy rule templates (see section B.6) is extremely general and, therefore, easily adhered to when writing new policy rules.

Referring to section B.4, the first step performed by the procedure `Configuration` is to call a procedure (`Get_Property_Group_Members_$configItem`) that examines the communications plan (i.e., the MIB cache), and extracts all the network elements that are members of property groups related to the item being configured. `Get_Property_Group_Members_$configItem` is a different procedure for each configuration item, therefore, if one extends the functionality of the AMA by adding a new item to the configurable items list, an additional procedure must be written. In practice, the `Get_Property_Group_Members_$configItem` procedures are similar to each other and, therefore, are easily written. This step, and the subsequent steps taken by procedure `Configuration` are summarized as follows:

a. Extract from the communications plan the network elements that are members of property groups related to the item being configured;

b. Read the appropriate properties data base file that contains lists of properties indexed by property groups;

c. Create lists of network elements (or members) indexed by property;

d. Read the policy rule templates from the appropriate policy rules file;

e. Form completed policy rules by expanding the policy rule templates in combination with the lists of members indexed by property;

f. Call a procedure (`Mdp_$configItem`) that will implement the policy rules.

Once again, the procedure `Mdp_$configItem` is a different procedure for each configuration item, therefore, if one extends the functionality of the AMA by adding a new item to the configurable items list, an additional procedure must be written.

The next four sections are a summary of how the four configurable items currently supported by the prototype are processed by the MDP. These sections will provide examples clarifying the operation of the MDP, as well as provide more insight into how the example of section 2.1 works.

2.2.2.2 INM SNMP Session Configuration

For the case of INM SNMP session configuration, in step "a" as described in the previous section, `Configuration` calls a procedure that reads the subordinate and superior INM tables (`subTable` and `supTable`), and returns lists of the unit's subordinate and superior INMs indexed by the values of `subInmType` and `supInmType` (which in this case define the property groups). Following along with steps b through f described in the previous section, the properties data base file (section B.5) is read, lists of members indexed by property are formed, the policy rules file (section B.6) is read, the lists of members indexed by property are combined with the policy rules templates, and finally, an expanded set of policy rules are passed to the procedure `Mdp_InmSnmpSession` (the `Mdp_InmSnmpSession` code is relatively long and specialized and, therefore, not included in appendix B).

Examination of the policy rules file reveals that for the case of INM SNMP session configuration, there are three types of policy templates: one for configuring the SNMP session handles themselves; one for configuring SNMP polling (i.e., periodic queries) between INMs; and one for configuring SNMP informs (i.e., unsolicited fault-driven messages) between INMs. The first policy template in section B.6 is for setting up an SNMP session between INMs. It indicates that for each of the local unit's primary superior ROCs, `Mdp_InmSnmpSession` will create an SNMP agent handle using the max-access context to answer requests from the appropriate ROC IP addresses. The sixth policy template in section B.6 is similar, but results in SNMP agent handles with context readOnly for secondary superior ROCs. Similarly, the 14th policy template results in the setting up of SNMP manager session handles for all the local units subordinate primary LOCs.

Polling processes are set up by templates like the 15th policy template in section B.6. When expanded and processed by `Mdp_InmSnmpSession`, this template sets up a polling process for all the local unit's subordinate primary LOCs. The arguments in the policy template indicate that the (highest level) operational status of the subordinate should be polled every 60 seconds.

Finally, the second policy template in section B.6 results in the set up of inform requests. Specifically, SNMP inform requests are generated from the local unit to its primary superior ROCs when the mission operational status rises to the `c-3` state or higher, or falls to the `c-1` state. In processing the inform policy rules, the procedure `Mdp_InmSnmpSession` populates the `inmAlarmTable`, `inmEventTable`, and `inmEventNotifyTable` which are members of the INM notification group in the AA MIB. These tables were adapted from the ideas presented in RFC1451 (reference 14), and hold the configuration information that controls the generation of inform requests. Note that rows in these MIB tables can be created and deleted by managers with appropriate access rights. The configuration set up by the MDP can, therefore, be modified on-the-fly by managers with proper authority.

In this example, it is seen that the communications plan identifies the hierarchy of INMs, the type of each INM, and their address. The policy rules dictate how, what, and when these INMs will communicate. By changing these policies, the manner in which INMs communicate can be modified.

2.2.2.3 MLM Configuration

As described in section 2.1, an MLM performs periodic polling of network elements, filtering the resulting responses from network elements, and filtering unrequested messages (i.e., SNMP traps) from network elements. Using the procedure described in section 2.2.2.1, and detailed for the case of SNMP INM configuration in section 2.2.2.2, a set of expanded policy rules specific to the configuration of the current mission are passed to the procedure `Mdp_Mlm`. For example, the communications plan might indicate the location of a channel access protocol (CAP) router interface unit (CRIU) and the location and number of CAPs connected to it. A policy rule template, for example:

```
c:criu-monitoring p:cap-monitoring \
POLL_CONFIG SNMP:c:1801:aicsInm:1.3.6.1.4.1.1738.2.300.3.1.1.8.p ADD:40:5
```

might be included in the MLM policy rules file which, when expanded, results in a policy rule sent to the `Mdp_Mlm` to configure the Mlm to poll the CRIU for the number of source quenches each of the CAPs have sent, and if more than a certain amount have been sent in a certain period of time, to then forward a trap to the local INM.

There are many different choices available to implement the MLM function. The MLM capability could be built directly into the AMA, or a variety of COTS products of varying of capabilities could be employed. In the current prototype Computer Associates' Domain Manager (reference 15), a COTS MLM program, is used to fulfill the polling and filtering requirement. The procedure `Mdp_Mlm` checks the value `unitInmModuleName` corresponding to the MLM, and calls a routine that converts the policy statements (internal format), into the configuration in the format specific to the MLM in use. This routine then (re)starts the MLM with the new configuration information.

This design is shown in figure 2. The MDP is effectively split into two parts. The front-end determines the desired MLM configuration as dictated by merging the communications plan (as stored in the AA MIB cache) and the policy rule templates, and represent this resulting configuration in an internal format (i.e., the expanded policy rules). The back-end converts the internal format into configuration files for the specific MLM in use and (re)starts the MLM. This design is such that by changing only the output side of the MDP, the choice of COTS mid-level managers can be changed.

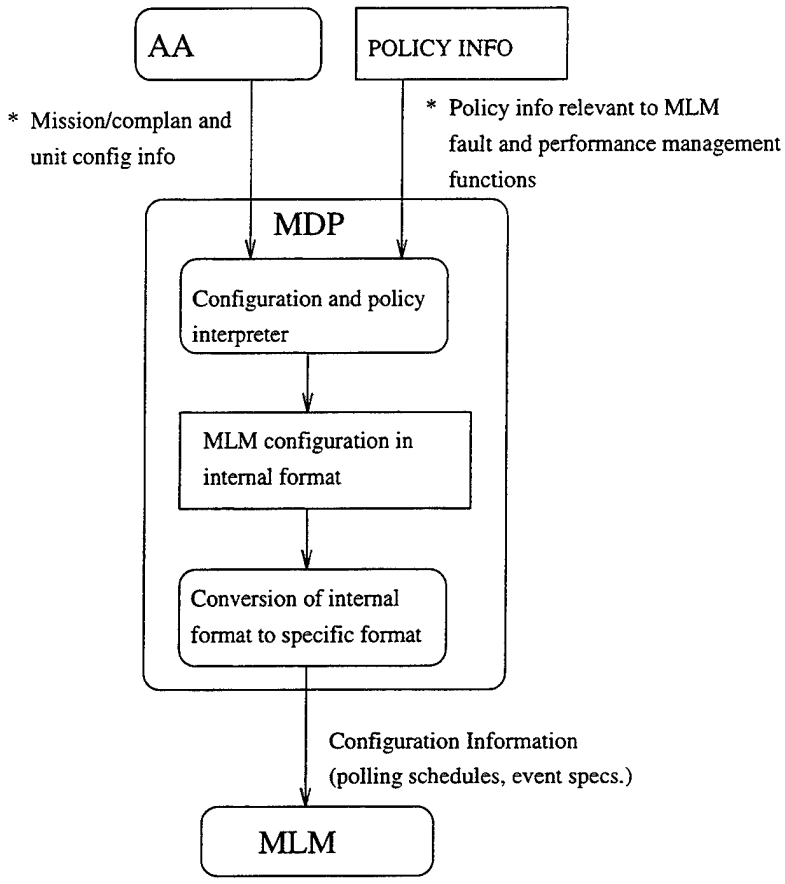


Figure 2. Diagram of the MDP for the MLM.

2.2.2.4 Operational Status Dependency Configuration

In the previous section, the monitoring of local assets and resulting generation of traps by the MLM to the local INM was discussed. In addition, in section 2.2.2.2, the configuration enabling fault-driven inform requests from subordinate INMs to superior INMs was discussed. In between these two processes, the local INM needs to process the traps it receives from the MLM to determine what inform requests, if any, should be sent to superiors. Much like the MLM is programmed to correlate and filter the information it receives so as to present the local INM with more useful information, the INM must also correlate and filter information it receives to determine what messages should be sent to superior INMs.

MLMs use a variety of methods to enable their correlation and filtering capability. In the prototype, the INM method of correlation and filtering is incorporated into the AA MIB, thereby providing a concise definition of the correlation and filtering process, and standards-based access to this information. The use of the `inmAlarmTable`, `inmEventTable`, and `inmEventNotifyTable` to facilitate generation of inform requests from subordinate to superior INMs was discussed in section 2.2.2.2. To facilitate correlation of aggregate operational status information, indexing of ob-

jects in the MIB is such that a hierarchy of operational status objects exists. The `inmOpStatusDependencyTable` is provided to define the necessary details of the dependence of operational status objects. Once again, these details are initialized by the MDP using steps a through f as described in section 2.2.2.1. The rows of `inmOpStatusDependencyTable` can be created and deleted; therefore, the configuration set up by the MDP can be modified on the fly by managers with proper access rights.

As an example, these policy rules might create a setup where:

- The operational status of the mission depends on some weighted average of the operational statuses of all the communications services utilized for the mission;
- Each communications service depends on some weighted average of operational statuses of the transmission resources they utilize;
- Each transmission resource depends on some weighted average of the operational statuses of the equipment which makes up the resource; etc.

As a result, when, for instance, the MLM detects that a CAP has sent too many source quenches, the trap sent to the AMA will cause the operational status of the CAP to change. This will cause the AMA to redetermine the operational status of the transmission resource that uses that CAP, the communications services that use that transmission resource, etc., all the way up to the overall operational status of the mission. The policy rules would likely be set up such that only when the higher level aggregate operational status objects change to more critical levels will inform requests be sent to superior INMs (see section 2.2.2.2).

A possible deficiency of the current implementation is that the algorithm used for computing the value of the aggregate operational status variables does not incorporate temporal memory, but instead always calculates a (policy-governed) weighted average based on the instantaneous value of other statuses. Experience has shown that introducing memory into the algorithm by the use of "state machines" is a useful tool in assessing the status of a network. As a result, this is a common feature in available MLMs (e.g., Domain Manager), and can, therefore, be utilized by the INM for processing management information at the MLM level. As the AMA only computes aggregate status information, it is not clear that state machines would be as useful as they are when applied directly to network objects. For this reason, and that it would entail adding complexity to the AMA, this functionality was not included in the current prototype.

2.2.2.5 RMON Configuration

The last configuration item supported by the current version of the INM prototype is RMON. An RMON probe listens promiscuously on a LAN, and can be programmed to collect information about the packets that are transmitted. There are two RMON standards, the RMON-1 (reference 16) standard, and RMON-2 (reference 17), which is just now reaching the culmination of the standardization process. RMON-1 includes the capability to access link layer (i.e., ethernet layer) statistics quickly. It also includes the capability to filter and selectively capture packets, thereby enabling analysis of higher network layers in a less convenient manner. The RMON-2 standard incorporates quick access to statistics all the way up to the application layer. Current COTS RMON probes support RMON-1, and include support for features of RMON-2, but not necessarily in a way that conforms to the RMON-2 standard. It is anticipated that COTS RMON-2 conformant probes will be available within the next year.

In the context of the Advanced Digical Network System (ADNS), monitoring network layer statistics (as opposed to link layer statistics) would be a useful INM capability because it allows one to see beyond a router. In the absence of COTS RMON-2 products, an agent supporting parts of the RMON-2 MIB was developed under the AICS program. This RMON-2 "simulation" agent configures an RMON-1 probe, retrieves packets previously captured by the RMON-1 probe, analyzes the packets, and populates the RMON-2 MIB tables holding information detailing the top bandwidth network layer conversations. A list of the top bandwidth network conversations can then be retrieved via SNMP in the same manner as if the RMON-2 agent were actually an RMON-2 probe.

In the AMA prototype, the procedure `Config_Check` examines the `unitInmModuleTable`, and if the local unit has an RMON-2 "simulation" agent (and an RMON-1 probe), the agent is started. In addition, a menu item is added to the AMA Display to view the information made available by this agent.

When COTS RMON-2 probes become available, the RMON-2 "simulation" agent will become obsolete for INMs equipped an RMON-2 probe. For the AMA to support the RMON-2 probes, a new stanza would be added to the procedure `Config_Check` (to check if the local unit has an RMON-2 probe), and an additional procedure called from the MDP would be written. Because the RMON-2 "simulation" agent has the same interface as an actual RMON-2 probe, the additional MDP procedure would be a simple addition.

2.2.3 AMA Display

Figure 3 shows an example of the current AMA prototype display. Included in the upper right of the figure is a window belonging to OV-NNM showing symbols representing various devices on a particular LAN. On the left hand side of this window is an oval symbol representing the INM and indicating its top level status. In the upper

right of the OV-NNM window is the menu item used to launch the AMA. The other windows in the figure belong to the AMA. The current display is not refined, but demonstrates the basic function of the AMA display. In the upper right of the AMA's main window is a symbol for the overall mission operational status labelled "OP-PLAN STATUS." This status is tracked by the symbol representing the INM on the left of the OV-NNM window. Below the symbol for the mission operational status are symbols for the aggregate operational statuses of the applications utilized by the current mission. By clicking on these symbols, the aggregate statuses of the network elements on which the user applications depend are revealed. By continuing to click on the aggregate operational status symbols as they appear, the operational status of more detailed aspects of the network is revealed. The AMA presents information in the context of the unit's current mission, whereas the SNMP management platform program presents a network topology that may appear abstract to all but highly trained operators. As rudimentary as figure 3 is, it demonstrates the difference in the type of information that is displayed by an SNMP management platform program (in this case OV-NNM) and the AMA.

2.3 Network Standards and COTS Product Integration

It is the current consensus that using accepted network standards and taking advantage of COTS products in the design of Navy systems is a cost-effective strategy. There are several aspects of the INM prototype where this principal has been incorporated into the design:

- The design of the prototype INM is such that when COTS network management products fulfill INM requirements, they can be integrated into the INM. As the technical problems of integration of particular products have to be solved on a case-by-case basis, the prototype INM framework was made flexible. In section 2.2, integration of an SNMP Management Platform Program, MLM, and RMON probe into the prototype INM was demonstrated.
- The requirements of one unit's INM might be different than those of another unit's, therefore, the AMA was designed for flexibility in the choice of the COTS products that are utilized. With the current rapid evolution of COTS network management software products and protocols, this flexibility is particularly important as it reduces the need for commitment to a particular vendor's products and methodology, and the constraints this commitment entails.
- The AMA was developed using an off-the-shelf toolkit (in this case, a public domain toolkit, not a commercial one). The toolkit is based on a widely known, powerful, well documented, portable scripting language (Tcl/Tk) (as opposed to unique, limited, little-known languages that some COTS toolkits employ). Although this is not an endorsement of the partic-

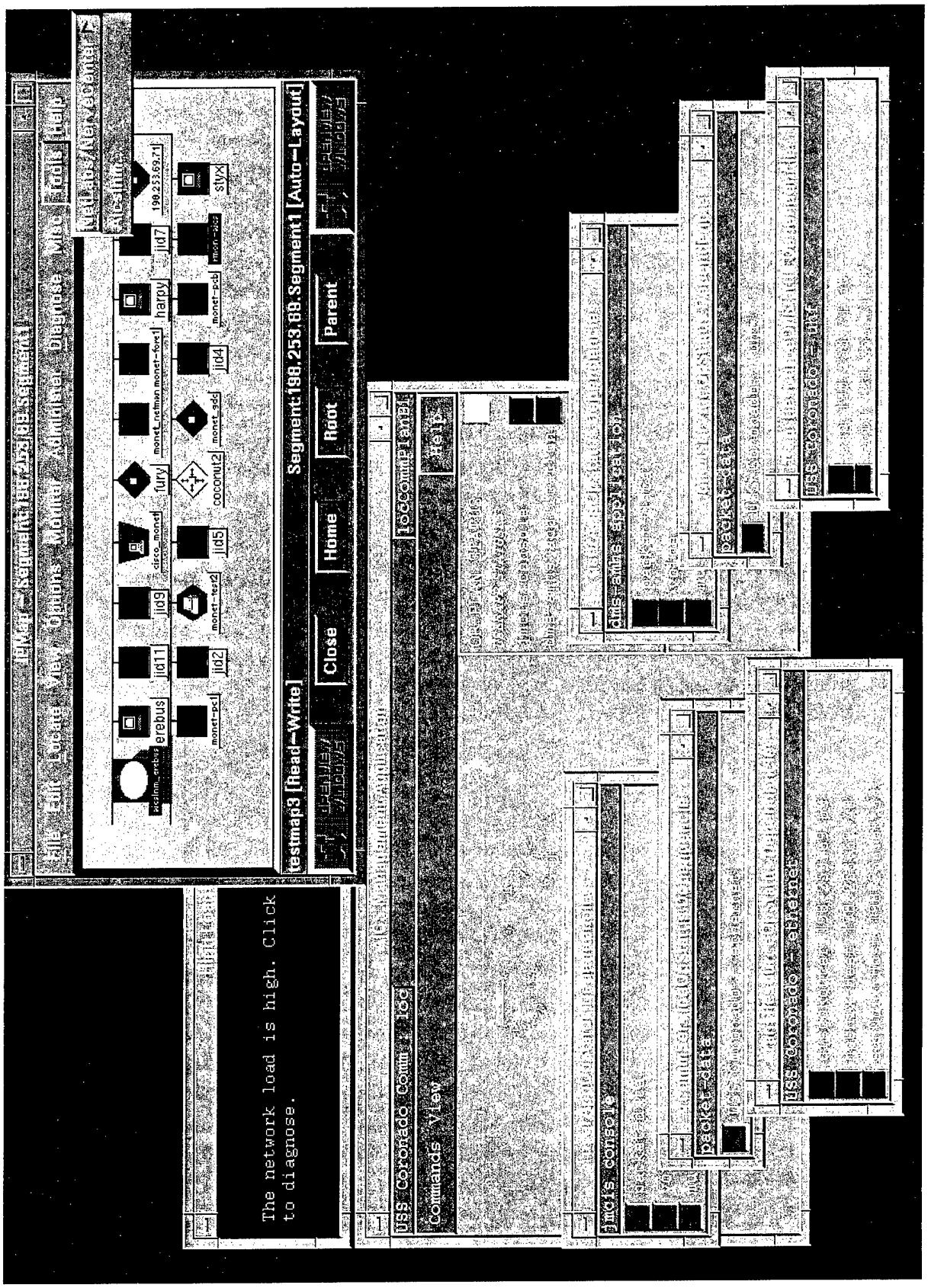


Figure 3. Example of the INM display.

ular toolkit used, it is meant to emphasize that when a toolkit is required, using a toolkit with these qualities is advantageous not only for initial development, but for flexibility, serviceability, and extensibility over the course of the system's life cycle.

- In the prototype, some of the AMA's functionality is incorporated into the AA MIB, thereby providing a definition of these functionalities using accepted network management standards. It is envisioned that in future versions of the AMA, its functionality will be more thoroughly defined in the AA MIB.

In this report, the only COTS network management programs discussed are the SNMP management platform program and the MLM. There are a wide variety of other COTS programs serving a range of functions— physical network management, trouble ticketing, and the gamut of management programs for specialized network devices, to name a few. Presently, the networks that are the primary target of the AICS program are not mature enough to make a discussion of these more specialized types of management programs worthwhile. The integration of the COTS SNMP management platform program and MLM functionalities into the INM fulfills INM requirements that will be common to many units. The next two sections discuss these two INM components.

2.3.1 COTS SNMP Management Platform Program Integration

Current typical COTS SNMP management platform programs provide a variety of required services, including a display showing the topology and status of the network, menu-driven methods to customize the appearance of the network topology display, automatic monitoring of the network using ICMP packets, an SNMP protocol engine, network auto-discovery, a general purpose MIB browser, customizable menu items for retrieving specific MIB objects, an SNMP trap daemon and methods to bind actions to received traps, and an API for third party program development. Differentiating between different COTS SNMP management platform programs is problematic. The division between what functionality is built directly into the SNMP management platform program and what is left for third party development is up to the discretion of each vendor. Because these products are relatively new, the functionality provided in each vendor's product and what third party add-ons are available can change considerably from year to year. In fact, the networks that these programs are designed to manage are also quickly evolving (e.g., the emergence of virtual LANs); therefore, one might expect that the COTS SNMP management platform of today will bear only a slight resemblance to ones several years from now. The INM prototype described in this document does not make choices between these products, but instead emphasizes that the design should be committed to accepted network standards and remain flexible with respect to use of specific products whenever possible.

It should be emphasized that not all units will require the a fully functional COTS

SNMP network manager platform program. For instance, those units that have few network assets will not need such a program. In addition, those units that do not have a knowledgeable network technician would also not be able to make use of such a program. The default topology map as displayed after an auto-discovery process will only have meaning to a skilled operator. Even a customized display, pre-configured from knowledge of the mission communications plan, may not result in a display useful to the available personnel. If the configuration of the SNMP network manager platform program needs to be changed, the operator would have to understand both the underlying network technology and be skilled at using the particular SNMP network manager platform program graphical user interface (GUI). It is likely that such a level of expertise would not be available on many units.

For those units which require a COTS SNMP network manager platform program, a choice between the various ones that are available will have to be made. In the INM prototype, two different SNMP network manager platforms were employed, a commercial one, OV-NNM, and a public domain one, Tkined (part of the *Scotty* package). Because of the loose integration level between the SNMP network manager platform program and other INM components, it was relatively easy to use SNMP network manager platform programs interchangeably. This flexibility is, of course, lost when INM modules are more closely integrated with the SNMP network manager platform program. For instance, if an INM module uses the the SNMP network manager platform program API to utilize its protocol engine or access its topology data base, a commitment to the particular SNMP network manager platform program is made.

Besides HP OV-NNM, there are several other notable COTS SNMP network manager platform products available. Cabletron Systems Inc.'s Spectrum (reference 18) is probably the most innovative of the SNMP network manager platforms. Spectrum uses a technique wherein a model can be created for network objects that creates relationships between network objects. In essence, this methodology elegantly incorporates correlation and filtering functionalities into Spectrum, functions which in this document have been associated with an MLM, and which are handled by third party add-ons in most other SNMP network manager platforms. Spectrum uses a pure client-server distributed architecture, where the network managment displays are clients that connect to servers where the network management related databases reside and where network management functions are performed. Although more capabilities may be built into Spectrum than other SNMP network manager platforms, Spectrum's third party support is not as good as, for instance, OV-NNM. If third party support is an indication of acceptability, then OV-NNM might be considered the benchmark of SNMP network manager platforms. Both Sun Microsystems Inc. (Solstice SunNet Manager) (reference 19) and IBM (NetView) (reference 20) have product offerings that provide functionality simliar to OV-NNM. As stated above, these products are evolving at such a pace that, in the context of this report, more detailed comparisons between them would be of limited value.

2.3.2 COTS MLM Integration

The functionality of an MLM will likely be required by all LOCs. There is a variety of products and strategies that can be used to provide the functionality required of a particular unit's MLM. As discussed in section 2.2.2.3, Domain Manager was used in the INM prototype. Domain Manager was an appropriate choice for the prototype because monitoring of only a few network devices was required. Whether or not Domain Manager, or a product of similar capabilities, would be appropriate for a fieldable INM would depend on the requirements of the particular unit. The qualities that distinguish Domain Manager are as follows: it is low cost; it is designed to monitor a relatively small number of network devices; it is a stand-alone product in that it is not tied to any COTS network management platform; it is configurable via flat files; it runs as a set of background processes; and it provides basic monitoring and filtering capabilities.

For units that have larger networks, a program such as Seagate's Nerve Center might be appropriate. Nerve Center is an expensive, large program requiring significant computing resources and is meant to manage a relatively large number of network devices. It uses the methodology of properties and property groups discussed in section 2.2.2.1 to enable easier configuration of large networks. Nerve Center is closely integrated with OV-NNM. Although future versions may be developed that can be integrated with other COTS network manager platforms, presently using Nerve Center is a commitment to using OV-NNM. Nerve Center has a relatively elaborate GUI for configuration, although this GUI creates flat files that fully describe the configuration. Configuration of Nerve Center in accordance to the communications plan and policy rules could, therefore, be implemented in a way similar to that which was done with Domain Manager in the current INM prototype. Nerve Center can also be run as a stand-alone program (i.e., without OV-NNM) in an unattended mode, and the Nerve Center nodes can be arranged in a hierarchy. (When run as a stand-alone program, it is required that Nerve Center is running in the OV-NNM overlay configuration somewhere else on the network). Nerve Center and similar products are typically designed to operate in an interconnected ethernet LAN environment, not an internet of networks separated by low-bandwidth RF links. At times (e.g., startup) the various Nerve Center nodes in the management hierarchy may pass a significant amount of information to each other; therefore, for the Navy's application, a detailed understanding of the programs operation would be required to avoid saturation of RF links with network management information.

Another notable MLM is made by SNMP Research, Inc (reference 21). Through the use of set requests containing scripts, the SNMP Research, Inc. MLM is configurable via SNMP. The SNMP Research, Inc. MLM runs purely as background processes (i.e., in an unattented mode). In the framework of the AMA prototype, the back-end of the MDP could send the required SNMP requests to the (local or remote) MLM. For cases where a unit does not have an AMA, this is a particularly desirable feature

as remote configuration over the RF link would be done using SNMP. In addtion, SNMP Research Inc. products are some of the few available that currently support secure SNMP communications.

3. SUMMARY AND CONCLUSION

This report documents the INM prototype developement and integration effort, part of the FY96 AICS program. The ultimate vision for AICS is a highly automated network management system where its functions are implemented using standards-based management protocols and may be carried out in a virtually unattended mode. The prototype INM gives a preview of this ultimate vision and starts AICS network management development in a direction that can evolve with new technology.

The AICS architecture, as descibed in the NMA and SSS, describes a hierarchy of INMs, where the INMs at the lowest level of the hierarchy (LOCs) are responsible for the communications/network assets under their purview. The AICS architecture is primarily targeted for an environment where INMs are commonly separated from other INMs by RF links. The main INM prototype development work was the AMA, a network management application that, within the framework of the AICS architecture, is meant to provide several Navy-specific functions and facilitate integration of other (COTS) network management programs into the INM. Much of the AMA's functionality is described through the AA MIB, thereby providing a standards-based definition of this functionality. If future versions of the AA MIB become more complete and robust, the AA MIB could serve to define the AMA interface concisely, possibly enabling independent development of inter-operable AMAs.

The communications plan, cast in the form of a set of MIB objects, and a set of policy rules applicable to each of the INM configuration management tasks are the inputs to the AMA. At the initiation of a new mission, the communications plan is loaded, the configurations tasks are performed, and monitoring of the network and reporting of faults and status begins, all in accordance with applicable policy rules.

In the current prototype, four configuration tasks are performed by the INM, although, as was discussed in section 2.2, the design is extensible so that other configuration tasks, and the policy rules associated with them, can be added. For instance, if a specialized management application that controls Navy legacy RF gear (i.e., the Communications Automation Manager[1]) is incorporated into the INM, then a configuration item and associated policy information for this application would be added to the AMA. The result would be that the AMA would send to the Communications Automation Manager the information (based on the communications plan and policy rules) the Communications Automation Manager requires to perform its tasks.

Because units with different network assets and different requirements will have INMs, the prototype design is such that all INMs need not be alike. The previous sections described several modules that might be components of an INM. A unit's network environment, required tasks, and available operators would determine its INM configuration. For example, one unit may require an INM with a COTS SNMP management platform, while others may not, while one INM might use one type of mid-level manager, and another INM a different one.

The prototype INM and the ideas it demonstrates are clearly aimed at a next generation network management system. If one has an operational network that lacks proper network management tools, and if one is interested in implementing an immediate solution, then many of the ideas presented in this document would be of limited value. The strategy one would take in such a situation would hardly begin to touch upon such issues as automated management based on Navy communications plans and policy rules, as is addressed in the INM prototype.

The INM prototype is by no means a finished product, but is meant to present some useful ideas in a working prototype and provide a good starting point for future development. The following are some of the specific topics of future work:

- The design of the prototype depends both on machine-readable communication plans formatted for use by the INM, and a MIB capable of holding the essence of said communication plans. The generation of machine-readable communications plans is beyond the scope of the FY96 AICS program, though once these machine readable communication plans are available, they would have to be formated for use by the INM, and neccessary modifications to the AA MIB would have to be made.

- The prototype uses SNMPv2-USEC authentication for INM to INM communications, but the USEC authentication keys are presently handled in a perfunctory manner. As with any system that uses encryption, a carefully thought out procedure is required for distribution and management of encryption keys.

- The prototype does not provide a user friendly interface for writing or modifying the policy data base files. Such an interface would be necessary if operators in the field are allowed access to the policy data base files.

- While, the current AMA display served its intended purpose for the prototype, it lacks functionality necessary for a fieldable product. Necessary additions include an interface specially designed for browsing the AA MIBs of the local and subordinate units. More general changes to the AMA display might also be considered. For instance, due to the pervasiveness of Hypertext Markup Language (HTML) viewers (used by World Wide Web (WWW) clients), the development of a Hypertext Transport Protocol (HTTP) server as a means of providing AMA information to operators via WWW clients has important advantages. By utilizing WWW clients as an interface to access AMA information, the need for development of a custom GUI, and the need for operators to learn how to use a custom GUI, are eliminated. Such HTTP servers would have to be designed such that HTTP is used only within a local unit, and SNMP is used over low-bandwidth links.

- In section 2.3 it was stated that the technical problems of COTS product

integration would be solved on a case-by-case basis. As there is little means to control the type of interfaces and configuration methods used by COTS products, let alone their compatibility with each other, the issue of integration will persist. Development of generalized ways of handling the integration problem is an area of interest in the commercial community. Exploiting new integration methods will continue to be an area of interest for Navy network management systems.

The networks for which the AICS effort is intended are only now beginning to be fielded. As a result, there has been little opportunity to study the problems that occur in the operational environment. Typically, elegant network management solutions result from observations by trained technicians of the problems that occur during operation of specific networks, followed by iterative applications of more satisfying solutions. To reach its fruition, an advanced network management system such as the one outlined in the NMA and SSS and prototyped in this document will require implementation in a testbed simlar to the targeted network environment, followed by an incremental integration into the operational network.

4. REFERENCES

1. MITRE Corporation. "Automated Integrated Communications System Network Management Architecture," and "Automated Integrated Communications System Network Management System/Segment Specifications," Prepared for Naval Command, Control and Ocean Surveillance Center RDT&E Division, Code 82, San Diego, CA 92152-50001.
   Available via ftp (see directory ftp://fury.nosc.mil/pub/AICS).

2. K. McCloghrie. 1996. "An Administrative Structure for SNMPv2," RFC1909.

3. G. Waters. 1996. "User-Based Security Model for SNMPv2," RFC1910.

4. Hewlett Packard Corporation, Cupertino, CA. Information available from the following URL: http://www.hp.com:80/nsmd/ov/main.html

5. J.K. Ousterhout. 1994. *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Co., Boston, MA.

6. Jürgen Schönwälder. Scotty - Tcl Extensions for Network Management Applications. University of Twente, Postbus 217, 7500 AE Enschede, Netherlands. Information available from the following URL:
   http://wwwsnmp.cs.utwente.nl/ schoenw/scotty/

7. I.N., Krishnan and S.G. Bradley. 1994. "AN/SSQ-33(V)3 Control and Management Segment Information Reference Model (IRM)," Document number WN 94W0000151, The MITRE Corporation, McLean, VA.

8. J.D. Day and H. Zimmerman. 1983. "OSI Reference Model," *Proceedings of the IEEE*, vol. 71, pp. 1334-1340.

9. Complete AICS Management Application code is available from authors of this report upon request.

10. L. Lewis. 1996. "Implementing Policy in Enterprise Networks," *IEEE Communications Magazine (Jan.)*, pp. 50-55.

11. M. Sloman. 1994. "Policy Driven Management for Distributed Systems," *Journal of Network and Systems Management*, vol. 2, no. 4.

12. M. Sloman, J. Magee, K. Twidle, and J. Kramer. 1993. "An Architecture for Managing Distributed Systems," *Proceedings of Fourth IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 40-46.

13. Seagate Enterprise Management Software, Cupertino, CA. Information available from the following URL:
    http://www.sems.com/Products/West/nervecenter/Nervecenter.html

14. J. Case, K. McCloghrie, M. Rose, S. Waldbusser. 1993. "Manager-to-Manager Management Information Base," RFC1451.

15. Computer Associates, Inc., One Computer Associates Plaza, Islandia, NY 11788-7000.

16. S. Waldbusser. 1995. "Remote Network Monitoring Management Information Base," RFC1757.

17. S. Waldbusser. 1996. "Remote Network Monitoring Management Information Base," working draft IETF Internet Draft draft-ietf-rmonmib-rmonmib-03.txt.

18. Cabletron Systems, Inc., 36 Industrial Way, Rochester, NH 03867. Information available from the following URL: http://www.cabletron.com/spectrum

19. Sun Microsystems, Inc., 2550 Garcia Ave., Mtn. View, CA 94043-1100. Information available from the following URL: http://www.sun.com/solstice/em-products/network/ent.man.html

20. IBM Corporation, Research Triangle Park, NC 27709. Information available from the following URL: http://networking.raleigh.ibm.com/netalpha.html.

21. SNMP Research International, Inc., 3001 Kimberlin Heights Road, Knoxville, TN 37920-9716. Information available from the following URL: http://www.snmp.com/

APPENDIX A: AICS MANAGEMENT APPLICATION AGENT MIB

```
AICS-Agent-MIB DEFINITIONS ::= BEGIN

        IMPORTS
            MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
            Integer32, Counter32, snmpModules
                FROM SNMPv2-SMI
            DisplayString, InstancePointer, RowStatus, TimeStamp
                FROM SNMPv2-TC
            MODULE-COMPLIANCE, OBJECT-GROUP
                FROM SNMPv2-CONF
            nrad, code827
                FROM NRaD-MIB;


--  ************************************************************************
        aics MODULE-IDENTITY
            LAST-UPDATED "9611180000Z"
            ORGANIZATION "NRaD, Code 827"
            CONTACT-INFO
                    "       Everett W. Jacobs

                            NCCOSC RDT&E DIV
                            Code D364
                            49590 Lassing Rd.
                            San Diego, CA 92152-6171

                        Tel: (619) 553-1614

                     E-mail: jacobs@nosc.mil"
            DESCRIPTION
                    "AICS Management Application Agent MIB module."
            ::= { code827 2 }

--  ************************************************************************
--
--      This is an experimental MIB used as the basis of the
--      AICS Management Application.  See NCCOSC RDT&E Div
--      Technical Report XXXX, "AICS Integrated Network Manager
--      Prototype Documentation" for general information about
--      the AICS Management Application.
--


        inm             OBJECT IDENTIFIER ::= {aics 1}

        mission         OBJECT IDENTIFIER ::= {inm 1}
        unit            OBJECT IDENTIFIER ::= {inm 2}
        inmNotification OBJECT IDENTIFIER ::= {inm 3}
```

```
    missionDef          OBJECT IDENTIFIER ::= {mission 1}
    missionCommPlan     OBJECT IDENTIFIER ::= {mission 2}
    missionSec          OBJECT IDENTIFIER ::= {mission 3}
    missionConfig       OBJECT IDENTIFIER ::= {mission 4}

    unitDef      OBJECT IDENTIFIER ::= {unit 1}
    unitNet      OBJECT IDENTIFIER ::= {unit 2}
    unitSec      OBJECT IDENTIFIER ::= {unit 3}
    unitOrg      OBJECT IDENTIFIER ::= {unit 4}
    unitSupport OBJECT IDENTIFIER ::= {unit 5}


-- The Mission Definition Group (missionDef)
    --
    -- A collection of objects which provide a high level
    -- description of the current mission that the unit
    -- is on.


    missionId      OBJECT-TYPE
        SYNTAX      Integer32
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "A integer which uniquely identifies the mission.  Encoded
              in this integer is the identification of which commPlan
              is to be loaded from the commPlan database."
        ::= { missionDef 1 }

    missionDescription  OBJECT-TYPE
        SYNTAX      DisplayString
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "A description of the mission."
        ::= { missionDef 2 }

    missionBeginTime OBJECT-TYPE
        SYNTAX      DisplayString
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
                   "The time the mission begins.  This time will be
                    with the format YYYY:MM:DD:HH:MM:SS, where YYYY
                    is the year, MM is the month, DD is the day, HH
                    is the hour, MM is the minute, and SS is the
                    second.  In next version this should be changed
                    to UTC Time format."
        ::= { missionDef 3 }

    missionEndTime OBJECT-TYPE
```

```
    SYNTAX      DisplayString
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
                "The time the mission ends.  This time will be
                 with the format YYYY:MM:DD:HH:MM:SS, where YYYY
                 is the year, MM is the month, DD is the day, HH
                 is the hour, MM is the minute, and SS is the
                 second.  In next version this should be changed
                 to UTC Time format."
    ::= { missionDef 4 }

nextMissionId     OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-write
    STATUS      current
    DESCRIPTION
         "A integer which uniquely identifies the next mission.
          Encoded in this integer is the identification of which
          commPlan is to be loaded from the commPlan database."
    ::= { missionDef 5 }


nextMissionBeginTime OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS      read-write
    STATUS      current
    DESCRIPTION
                "The time the next mission is scheduled to begin.
                 This time will have format YYYY:MM:DD:HH:MM:SS,
                 where YYYY is the year, MM is the month, DD is the
                 day, HH is the hour, MM is the minute, and SS is
                 the second.  In next version this should be changed
                 to UTC Time format."
    ::= { missionDef 6 }

nextMissionEndTime OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS      read-write
    STATUS      current
    DESCRIPTION
                "The time the next mission is scheduled to end.
                 This time will have format YYYY:MM:DD:HH:MM:SS,
                 where YYYY is the year, MM is the month, DD is the
                 day, HH is the hour, MM is the minute, and SS is
                 the second.  In next version this should be changed
                 to UTC Time format."
    ::= { missionDef 7 }

missionOpStatus   OBJECT-TYPE
    SYNTAX      INTEGER {
                     unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
```

```
                    down(5), failed(6)
                }
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
         "A integer which identifies the operational status of
          the local unit's  network/comm systems which are
          participating in the current mission.  This status is
          the highest level aggregate status for the unit.  This
          status will be indicated on superior INMs by the color
          of the symbol for this INM."
    ::= { missionDef 8 }

missionAdminStatus   OBJECT-TYPE
    SYNTAX      INTEGER {
                    running(1), reconfigure(2)
                }
    MAX-ACCESS      read-write
    STATUS      current
    DESCRIPTION
         "A integer which identifies the administrative status of
          the local unit's  network/comm systems which are
          participating in the current mission.  If
          missionAdminStatus is set by a higher level INM from
          running to reconfigure,
          and if nextMissionBeginTime has been reached,
          missionOpStatus will be changed to reconfiguring, and the
          process to change to he next mission will commence.
          After the reconfiguration procedure has finished,
          missionAdminStatus will be changed to running, and
          missionOpStatus will be changed from reconfiguring to
          the appropriate value."
    ::= { missionDef 9  }

supTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF SupEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
            "A list that identifies and describes the
             units that are superior to the local
             unit."
    ::= { missionDef 10 }

supEntry OBJECT-TYPE
    SYNTAX      SupEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
            "A set of parameters to identify and describe
             a unit that is superior to the local unit."
    INDEX       { supIndex }
```

```
    ::= { supTable 1 }

SupEntry ::= SEQUENCE {
    supIndex        INTEGER,
    supId           Integer32,
    supType         INTEGER,
    supDescription  DisplayString,
    supInmType      INTEGER,
    supInmIpAddress IpAddress,
    supInmPort      Integer32,
    supOpStatus     INTEGER
}

supIndex OBJECT-TYPE
    SYNTAX     INTEGER (1..65535)
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "An index that uniquely identifies an entry in the
            supTable.  "
    ::= { supEntry 1 }

supId  OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS     read-only
    STATUS     current
    DESCRIPTION
            "An identification code that uniquely identifies
             the superior unit."
    ::= { supEntry 2 }

supType   OBJECT-TYPE
    SYNTAX     INTEGER {
                    carrier(1), destroyer(2), submarine(3),
                    battleship(4), flagship(5), cruiser(6),
                    navy-base(7)
               }
    MAX-ACCESS     read-only
    STATUS     current
    DESCRIPTION
         "A integer which identifies the superior unit type."
    ::= { supEntry 3 }

supDescription OBJECT-TYPE
    SYNTAX     DisplayString (SIZE (0..127))
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "A comment describing the superior unit."
    ::= { supEntry 4 }

supInmType   OBJECT-TYPE
```

```
        SYNTAX      INTEGER {
                     none(1), primary-roc(2), secondary-roc(3),
                     other-roc(4), primary-noc(5), secondary-noc(6),
                     other-noc(7)
                    }
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "If applicable, an integer which identifies the superior
              unit INM type."
        ::= { supEntry 5 }

    supInmIpAddress OBJECT-TYPE
        SYNTAX  IpAddress
        MAX-ACCESS  read-only
        STATUS  current
        DESCRIPTION
                "If applicable, the IP address of the INM on the
                 superior unit."
        ::= { supEntry 6 }

    supInmPort   OBJECT-TYPE
        SYNTAX      Integer32
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "If applicable, the port to send communications to the INM
              on the superior unit."
        ::= { supEntry 7 }

    supOpStatus   OBJECT-TYPE
        SYNTAX      INTEGER {
                         unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                         down(5), failed(6)
                    }
        MAX-ACCESS      read-write
        STATUS      current
        DESCRIPTION
             "A integer which identifies the operational status of
              the superior unit's  network/comm systems which are
              participating in the current mission.  This status is
              the highest level aggregate status for the superior
              unit.  This status variable is not controlled by the
              local unit, but is writable by the superior to provide a
              means of pushing information down to a subordinate."
        ::= { supEntry 8 }


-- End of supTable
```

```
subTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF SubEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
            "A list that identifies and describes
             the units that are subordinate to the
             local unit."
    ::= { missionDef 11 }

subEntry OBJECT-TYPE
    SYNTAX      SubEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
            "A set of parameters to identify and describe
             a unit that is subordinate to the local unit."
    INDEX       { unitId, subIndex }
    ::= { subTable 1 }

SubEntry ::= SEQUENCE {
    subIndex         INTEGER,
    subId            Integer32,
    subType          INTEGER,
    subDescription   DisplayString,
    subInmType       INTEGER,
    subInmIpAddress  IpAddress,
    subInmPort       Integer32,
    subOpStatus      INTEGER,
    subAdminStatus   INTEGER
}

subIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..65535)
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "An index that uniquely identifies an entry in the
            subTable.  "
    ::= { subEntry 1 }

subId  OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "An identification code that uniquely identifies
             the subordinate unit."
    ::= { subEntry 2 }

subType   OBJECT-TYPE
    SYNTAX      INTEGER {
```

```
                    carrier(1), destroyer(2), submarine(3),
                    battleship(4), flagship(5), cruiser(6),
                    navy-base(7)
                }
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
         "A integer which identifies the subordinate unit type.
          Among other things, the value of subType will
          indicated if the unit has an INM."
    ::= { subEntry 3 }

subDescription OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..127))
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "A comment describing this sub entry."
    ::= { subEntry 4 }

subInmType   OBJECT-TYPE
    SYNTAX      INTEGER {
                 none(1), primary-loc(2), other-loc(3),
                 primary-roc(4), other-roc(5)
                }
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
         "A integer which identifies the subordinate unit INM type."
    ::= { subEntry 5 }


subInmIpAddress OBJECT-TYPE
    SYNTAX  IpAddress
    MAX-ACCESS  read-only
    STATUS  current
    DESCRIPTION
            "The IP address of the INM on the subordinate unit.
             If this subordinate  unit does not have an INM, the
             value of this variable is set to 000.000.000.000 ."
    ::= { subEntry 6 }

subInmPort   OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
         "If applicable, the port to send communications to the INM
          on the subordinate unit."
    ::= { subEntry 7 }

subOpStatus   OBJECT-TYPE
```

```
        SYNTAX      INTEGER {
                        unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                        down(5), failed(6)
                    }
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "A integer which identifies the operational status of
              the subordinate unit's  network/comm systems which are
              participating in the current mission.  This status is
              the highest level aggregate status for the subordinate
              unit.  This status will be indicated on the display of
              the superior INM by the color of a symbol for the
              subordinate."
        ::= { subEntry 8 }

    subAdminStatus   OBJECT-TYPE
        SYNTAX      INTEGER {
                        running(1), reconfigure(2)
                    }
        MAX-ACCESS      read-write
        STATUS      current
        DESCRIPTION
             "A integer which identifies the administrative status of
              the subordinate unit's  network/comm systems which are
              participating in the current mission."
        ::= { subEntry 9  }

-- End of subTable

-- End of missionDef group

-- The Mission Communication Plan Group (missionCommPlan)
    --
    -- A collection of objects which describe the operations plan
    -- and how units on a mission are to
    -- communicate.  There are two tables in the group, the
    -- user application table (userAppTable) and the
    -- communication service table (commServiceTable).  The
    -- user application table is for identifying the different
    -- user applications employed by the unit for the mission.
    -- The user applications
    -- are in general, associated with layer 7 of the OSI network
    -- model.  The communication service table is to identify the
    -- communication services that each user application requires,
    -- and to identify  the transmission
    -- resources each user-application/communication-service pair
    -- can utilize.  In the context of shipboard units, transmission
    -- resources are the particular RF resource. A user application
    -- can require more than
    -- one communication service.  A user-application/communication-
    -- service can utilize more than one transmission resource,
```

```
-- although generally a primary communication resource is identified.
-- In general, the communication service indicates  the quality
-- of service required by the application.   In reference to the
-- OSI seven layer model, layers 3 and 4 are typically most relevant
-- to communication services.   The transmission resources
-- in general refer to the RF communication equipment, and are
-- therefore more closely associated with layers 1 and 2 of the
-- OSI model.
--
-- On a LOC, all rows in these tables will be indexed by the local
-- unit.  On a higher level OC, there will be rows in these table
-- for subordinate units.
--

userAppTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF UserAppEntry
    MAX-ACCESS      not-accessible
    STATUS      current
    DESCRIPTION
            "A list that identifies and describes the
             user applications that are utilized on the
             current mission.  The user applications are
             in general associated with the application
             layer (layer 7) of the OSI network model."
    ::= { missionCommPlan 1 }

userAppEntry OBJECT-TYPE
    SYNTAX      UserAppEntry
    MAX-ACCESS      not-accessible
    STATUS      current
    DESCRIPTION
            "A set of parameters to identify and describe
             a user application."
    INDEX       { unitId, userAppIndex }
    ::= { userAppTable 1 }

UserAppEntry ::= SEQUENCE {
    userAppIndex            INTEGER,
    userAppType             INTEGER,
    userAppDescription      DisplayString,
    userAppHostIpAddress    IpAddress,
    userAppOpStatusWeight   Integer32,
    userAppOpStatus         INTEGER,
    userAppAdminStatus      INTEGER
}

  userAppIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..65535)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "An index that uniquely identifies an entry in
```

```
                userAppTable for a given unitId.  "
        ::= { userAppEntry 1 }

    userAppType OBJECT-TYPE
        SYNTAX      INTEGER  {
                        aps(1), andvt-stu-III(2), cds(3), ctaps(4),
                        dms-amhs(5), gccs(6), groupware(7), jdiss(8),
                        jmcis(9), navmacsII(10), ntcss(11), pixs(12),
                        saccs(13), s-tred(14), tacintelII(15),
                        tamps(16), tess(17), tty(18), vvfd-vtixs(19)
                    }
        MAX-ACCESS read-only
        STATUS      current
        DESCRIPTION
                "An integer that identifies the userApplication."
        ::= { userAppEntry 2 }


    userAppDescription OBJECT-TYPE
        SYNTAX      DisplayString (SIZE (0..100))
        MAX-ACCESS read-only
        STATUS      current
        DESCRIPTION
                "A description of the user application."
        ::= { userAppEntry 3 }

    userAppHostIpAddress OBJECT-TYPE
        SYNTAX  IpAddress
        MAX-ACCESS  read-only
        STATUS  current
        DESCRIPTION
                "If applicable, the IP address of the host on which
                 the user application is run."
        ::= { userAppEntry 4}

    userAppOpStatusWeight  OBJECT-TYPE
        SYNTAX      Integer32
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "Number of userAppOpStatus instances used to calculate
              the userAppOpStatus instance in this row. On
              a loc, this will be 1.  On a roc, it would be the
              number of subordinates that use this userApp."
        ::= { userAppEntry 5  }

    userAppOpStatus   OBJECT-TYPE
        SYNTAX      INTEGER {
                        unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                        down(5), failed(6)
                    }
        MAX-ACCESS      read-only
```

```
        STATUS      current
        DESCRIPTION
             "A integer which identifies the operational status of
              the user application.  This is an aggregate status of
              only the application program itself and the host it
              resides on, and not the communications resources it
              utilizes.   Failed is an indication that
              the application requires more than standard procedures
              to bring the application up."
        ::= { userAppEntry 6  }

    userAppAdminStatus   OBJECT-TYPE
        SYNTAX      INTEGER {
                         up(1), down(2)
                    }
        MAX-ACCESS      read-write
        STATUS      current
        DESCRIPTION
             "A integer which identifies the administrative status
              (i.e., the desired status) of the user application.
              This object is read-write if the unit identified by
              this row is the local unit, otherwise it is read-only.
              If the unit identified by this row is not the local unit
              then this object will be updated via status reports from
              the unit specified by this row."
        ::= { userAppEntry 7  }

-- End of userAppTable


    commServiceTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF CommServiceEntry
        MAX-ACCESS not-accessible
        STATUS      current
        DESCRIPTION
                "A table that includes all the userApp/commService
                 pairs for specified units, and identifies which
                 transmission resources each
                 userApp/commService utilizes.  The table is indexed
                 by userAppIndex, commServiceID, transResourceId
                 and unitId.  Before creating an entry in this table,
                 the management application should first check that
                 a row in the userAppTable corresponding to userAppIndex
                 and unitId, and a row in transResourceTable
                 corresponding to transResourceId and unitId exist.
                 If these rows do not exist, the agent should return an
                 error.

                 A userApp/commService
                 can utilize more than one transmission resource.  This
                 is because an application may need to communicate with
                 locations that can't be reached through the same
```

```
                transmission resource.  In addition, if one
                transmission resource is not available, a secondary
                one may be utilized.  In practice, a router will
                know that a userApp/commService can utilize more than
                one transmission resource, and it will choose the
                best one based on its routing protocol."

    ::= { missionCommPlan 2 }

commServiceEntry OBJECT-TYPE
    SYNTAX    CommServiceEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "A row that indicates the relationship between
             userApps, commServices, and transResources."
    INDEX {transResourceId, unitId, userAppIndex, commServiceType }
    ::= { commServiceTable 1 }

CommServiceEntry ::= SEQUENCE {
    commServiceType            INTEGER,
    commServicePriority        INTEGER,
    commServiceOpStatus        INTEGER,
    commServiceAdminStatus     INTEGER
}

  commServiceType OBJECT-TYPE
    SYNTAX     INTEGER  {
                 packet-data(1), voice(2), multimedia(3)
               }
    MAX-ACCESS     read-only
    STATUS     current
    DESCRIPTION
         "A communication service required by the application.
          A user application can require more than one communication
          service.  Communication service is an indication of the
          quality of service (and therefore often times the
          transport and network layers) required to support the
          user application.  In reference to the OSI seven layer
          network model, Communication service is therefore most
          closely associated with layers 3 and 4.  Currently
          only three communication services are defined."
    ::= { commServiceEntry 1 }

  commServicePriority OBJECT-TYPE
    SYNTAX     INTEGER  {
                 primary(1), secondary(2), backup(3)
               }
    MAX-ACCESS     read-create
    STATUS     current
    DESCRIPTION
          "Indicates if this is a primary transmission resource
```

```
            for this user-application/communication-service.  A
            user-application/communication-service can have more
            than one primary transmission resource because
            communication to locations that are not connected by
            the same transmission resource may be required."

    ::= { commServiceEntry 2 }

commServiceOpStatus   OBJECT-TYPE
    SYNTAX      INTEGER {
                    unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                    down(5), failed(6)
                }
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
         "A integer which identifies the aggregate operational
          status of the communication resources which are required
          by the user application on the specified unit.
          Failed is an indication that more than standard
          procedures are require to bring the communication
          resources up."
    ::= { commServiceEntry 3 }

commServiceAdminStatus   OBJECT-TYPE
    SYNTAX      INTEGER {
                    up(1), down(2), createAndGo(4), createAndWait(5),
                    destroy(6)
                }
    MAX-ACCESS      read-create
    STATUS      current
    DESCRIPTION
         "A integer which identifies the administrative status
          (i.e., the desired status) of the communications
          resources which are required by
          the user application on the specified unit.  A management
          station creates a row in this table by first setting
          the value of userAppResourceAdminStatus to createAndGo
          or createAndWait, and then setting commServiceType and
          primaryCommResource.  If using createAndWait, the agent
          would then set userAppResourceAdminStatus to down and
          userAppResourceOpStatus to down.  If using createAndGo,
          the AMA would then attempt the necessary configuration
          to bring the user application up.  If successful,
          userAppResourceAdminStatus and userAppResourceOpStatus
          would be set to up.  If unsuccessful, the agent will set
          userAppResourceAdminStatus to up and
          userAppResourceOpStatus to either down or failed depending
          on the problem."

    ::= { commServiceEntry 4 }
```

```
-- End of commServiceTable


--  End of mission Group

-- The Unit Definition Group (unitDef)
    --
    -- A collection of objects which provide a high level
    -- description of the unit associated with the AICS Agent.
    -- Currently, these objects identify the unit, and the
    -- configuration of the units INM.  This group might be
    -- expanded to include such information as  geographic location,
    -- and other objects which can be associated
    -- with the unit as a whole.


    unitId      OBJECT-TYPE
        SYNTAX      Integer32
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "A integer which uniquely identifies this (i.e., the local)
              unit."
        ::= { unitDef 1 }

    unitType    OBJECT-TYPE
        SYNTAX      INTEGER {
                         carrier(1), destroyer(2), canoe(3),
                         raft(4), flagship(5), mothership(6),
                         pacific-fleet(8), navy(9)
                    }
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "An integer which identifies the unit type."
        ::= { unitDef 2 }

    unitDescription OBJECT-TYPE
        SYNTAX      DisplayString (SIZE (0..100))
        MAX-ACCESS read-only
        STATUS      current
        DESCRIPTION
                "A description of the unit."
        ::= { unitDef 3 }

    unitInmType OBJECT-TYPE
        SYNTAX      INTEGER {
                        loc(1), roc(2), noc(3)
                    }
        MAX-ACCESS read-only
        STATUS      current
        DESCRIPTION
```

```
            "An integer that identifies the type of INM.  Although
             AMAs running in different modes (e.g., LOC, ROC, or
             NOC mode)  use an agent supporting the same MIB, the
             AMA's functions and display in these modes are quite
             different.  This object indicates the mode in which
             the AMA is running."
    ::= { unitDef 4}


unitInmModuleTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF UnitInmModuleEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "A list that identifies and describes
             the software modules of the INM on this unit."
    ::= { unitDef 5 }

unitInmModuleEntry OBJECT-TYPE
    SYNTAX     UnitInmModuleEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "A set of parameters to identify and describe
             an INM software module."
    INDEX      { unitId, unitInmModuleIndex }
    ::= { unitInmModuleTable 1 }

UnitInmModuleEntry ::= SEQUENCE {
    unitInmModuleIndex        INTEGER,
    unitInmModuleType         INTEGER,
    unitInmModuleName         DisplayString,
    unitInmModuleVersion      DisplayString,
    unitInmModulePort         Integer32,
    unitInmModuleOpStatus     INTEGER,
    unitInmModuleAdminStatus  INTEGER
}

unitInmModuleIndex OBJECT-TYPE
    SYNTAX     INTEGER (1..65535)
    MAX-ACCESS     read-only
    STATUS     current
    DESCRIPTION
            "An index that uniquely identifies an entry in the
            unitInm table.  "
    ::= { unitInmModuleEntry 1 }

unitInmModuleType OBJECT-TYPE
    SYNTAX     INTEGER {
                   nm(1), ama(2), aa(3), mlm(4), mdp(5), cam(6)
               }
```

```
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "An index that identifies the type of INM module."
    ::= { unitInmModuleEntry 2 }

unitInmModuleName OBJECT-TYPE
    SYNTAX     DisplayString (SIZE (0..40))
    MAX-ACCESS  read-only
    STATUS     current
    DESCRIPTION
            "The name of the INM module."
    ::= { unitInmModuleEntry 3 }

unitInmModuleVersion OBJECT-TYPE
    SYNTAX     DisplayString (SIZE (0..40))
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The version of the INM module."
    ::= { unitInmModuleEntry 4 }

unitInmModulePort OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The port number that would be used to communicate with
             this program, for instance, the port an MLM is listening
             to for SNMP requests.  The object should have value
             zero if there is no such port."
    ::= { unitInmModuleEntry 5 }

unitInmModuleOpStatus   OBJECT-TYPE
    SYNTAX     INTEGER {
                    unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                    down(5), failed(6)
               }
    MAX-ACCESS     read-write
    STATUS     current
    DESCRIPTION
         "A integer which identifies the operational status of
          the INM module identified by this row."
    ::= { unitInmModuleEntry 6  }

unitInmModuleAdminStatus   OBJECT-TYPE
    SYNTAX     INTEGER {
                    locked(1), unlocked(2)
               }
    MAX-ACCESS     read-write
    STATUS     current
    DESCRIPTION
```

```
            "A integer which identifies the administrative status of
             the INM module identified by this row."
        ::= { unitInmModuleEntry 7  }

    -- End of unitInmModuleTable



-- End unitDef group


-- The Unit Network Group (unitNet)

    -- A collection of objects that
    -- describe the configuration of the communication/network
    -- assets under the INM's control.  The layout of the  unit network
    -- group was influenced by the IRM presented
    -- in MITRE document WN 94W0000151.  The tables in this group
    -- describe a hierarchy made up of transmission resources,
    -- unit equipment, unit equipment ports, etc.
    -- In the IRM a rather general definition
    -- is given for a "network domain", where one type of network
    -- domain could be a particular RF network.   In this MIB,
    -- a less general  definition of "network domain" is used, and
    -- therefore what would have been the networkDomainTable is
    -- replace by the transResourceTable.  In the terminology of the
    -- IRM, the only allowed "network domains" are
    -- the aggregate equimpment from the link and physical layers,
    -- that take part in communication
    -- using a particular RF network (or transmission
    -- resource).  An entry in the transmission resourse table on a
    -- LOC will refer to the aggregate of all the equipment on the
    -- local unit that is part of the transmission resource.
    -- (e.g., all the equipment on a ship, from link and physical
    -- layers, used for communication on a particular RF network).
    -- On a LOC, the transResource table will typically only
    -- include entries for the transmission resources on
    -- the local unit, and possibly units which directly are connected
    -- to the local unit via links identified in the link table.
    -- On a LOC, the unit equipment and unit port
    -- tables will probably include complete information for all the
    -- local transmission resources in the transmission resource
    -- table.  On a ROC, typically there will
    -- be entries in the transmission resource table for each
    -- unit that is subordinate to the ROC, and entries indexed for
    -- to local unit which will hold aggregate status information
    -- over all the ROCs subordinates.  On a ROC the unit equipment,
    -- port and link tables may only have entries for a small subset
    -- of the units listed in its transmission resource table.
```

```
--


transResourceTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF TransResourceEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "The transmission resource table is a list of the
             RF network domains which are (in part or in whole)
             on or under control of this unit.  An RF network domain
             typically spans many units, with some subset of each
             unit's  assets belonging to the RF network domain."
    ::= { unitNet 1 }

transResourceEntry OBJECT-TYPE
    SYNTAX     TransResourceEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "A set of parameters to identify and describe
             a communication resource."
    INDEX      { transResourceId, unitId}
    ::= { transResourceTable 1 }

TransResourceEntry ::= SEQUENCE {
    transResourceId             Integer32,
    transResourceType           INTEGER,
    transResourceDescription    DisplayString,
    transResourceCntrlOrg       DisplayString,
    transResourceOpStatusWeight Integer32,
    transResourceOpStatus       INTEGER,
    transResourceAdminStatus    INTEGER
}


transResourceId OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "An integer that identifies the transmission resource."
    ::= { transResourceEntry 1 }

transResourceType OBJECT-TYPE
    SYNTAX     INTEGER {
                   uhf(1), vhf(2), los(3), satcom(4),
                   ethernet(5)
               }
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
```

```
            "An index that identifies the type of transmission
             resource."
    ::= { transResourceEntry 2 }


transResourceDescription OBJECT-TYPE
    SYNTAX     DisplayString (SIZE (0..100))
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "A description of the transmission resource."
    ::= { transResourceEntry 3 }

transResourceCntrlOrg OBJECT-TYPE
    SYNTAX     DisplayString (SIZE (0..100))
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
           "The organization that controls the transmission
            resource."
    ::= { transResourceEntry 4 }

transResourceOpStatusWeight  OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS     read-only
    STATUS     current
    DESCRIPTION
         "Number of transResourceOpStatus instances used to calculate
          the transResourceOpStatus instance in this row.
          On a LOC, this will be 1.  On a ROC, it would be the
          number of subordinates that use this transmission
          resource."
    ::= { transResourceEntry 5  }


transResourceOpStatus   OBJECT-TYPE
    SYNTAX     INTEGER {
                    unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                    down(5), failed(6)
               }
    MAX-ACCESS     read-write
    STATUS     current
    DESCRIPTION
         "A integer which identifies the operational status of
          the transmission resource.  This variable is write-able by
          a superior as a means of pushing information down
          to a subordinate."
    ::= { transResourceEntry 6  }

transResourceAdminStatus   OBJECT-TYPE
    SYNTAX     INTEGER {
                    locked(1), unlocked(2)
```

```
                }
        MAX-ACCESS      read-write
        STATUS      current
        DESCRIPTION
             "A integer which identifies the administrative status of
              the transmission resource."
        ::= { transResourceEntry 7  }


-- End transResourceTable


    unitEquipTable OBJECT-TYPE
                    SYNTAX
                    SEQUENCE OF UnitEquipEntry
                    MAX-ACCESS not-accessible
                    STATUS      current
                    DESCRIPTION
                   "The network unit equipment table is a list of the
                   equipment that is part of a particular unit
                   and part of a particular transmission resource.
                   For example this might be a list of all equipment on
                   a particular ship used to participate in
                   communications on a particular RF network.

                   A corresponding row in transResourceTable will exist
                   for rows in this table."


        ::= { unitNet 2 }

    unitEquipEntry OBJECT-TYPE
        SYNTAX      UnitEquipEntry
        MAX-ACCESS not-accessible
        STATUS      current
        DESCRIPTION
                "A set of parameters to identify and describe
                 equipment."
        INDEX       { transResourceId, unitId, unitEquipIndex }
        ::= { unitEquipTable 1  }

    UnitEquipEntry ::= SEQUENCE {
        unitEquipIndex          INTEGER,
        unitEquipId             DisplayString,
        unitEquipType           INTEGER,
        unitEquipIpAddress      IpAddress,
        unitEquipOpStatus       INTEGER,
        unitEquipAdminStatus    INTEGER
    }


    unitEquipIndex OBJECT-TYPE
```

```
    SYNTAX      INTEGER (0..65535)
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "An index that, coupled with transResourceId and
             unitId, uniquely identifies a row in this table."
    ::= { unitEquipEntry 1 }

 unitEquipId OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..100))
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "An unique  string identifier for the equipment."
    ::= { unitEquipEntry 2 }

 unitEquipType OBJECT-TYPE
    SYNTAX      INTEGER {
                    ciscoRouter(1), sunUnix-Host(2), hpUnix-Host(3),
                    rf-transmitter(4), criu(5), cap(6), rmon2-sim(7)
                }
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "An integer that identifies the type of equipment."
    ::= { unitEquipEntry 3 }

unitEquipIpAddress OBJECT-TYPE
    SYNTAX  IpAddress
    MAX-ACCESS  read-only
    STATUS  current
    DESCRIPTION
            "If applicable, the IP address of the equimpment."
    ::= { unitEquipEntry 4 }



 unitEquipOpStatus   OBJECT-TYPE
    SYNTAX      INTEGER {
                     unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                     down(5), failed(6)
                }
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
         "A integer which identifies the operational status of
          the equipment."
    ::= { unitEquipEntry 5  }

unitEquipAdminStatus   OBJECT-TYPE
    SYNTAX      INTEGER {
                     locked(1), unlocked(2)
```

```
                }
        MAX-ACCESS      read-write
        STATUS      current
        DESCRIPTION
             "A integer which identifies the administrative status of
              the equipment."
        ::= { unitEquipEntry 6  }


-- End unitEquipTable

    unitLogPortTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF UnitLogPortEntry
        MAX-ACCESS not-accessible
        STATUS      current
        DESCRIPTION
                "The logical port table is a list of the logical
                 ports on a particular piece of equipment that is part
                 of a particular unit and part of a particular
                 transmission resource.

                 A corresponding row in unitEquipTable will exist
                 for rows in this table."

        ::= { unitNet 3 }

    unitLogPortEntry OBJECT-TYPE
        SYNTAX      UnitLogPortEntry
        MAX-ACCESS not-accessible
        STATUS      current
        DESCRIPTION
                "A set of parameters to identify and describe
                 a logical port."
        INDEX       { transResourceId, unitId, unitEquipIndex,
                      unitLogPortId }
        ::= { unitLogPortTable 1  }

    UnitLogPortEntry ::= SEQUENCE {
        unitLogPortId           Integer32,
        unitLogPortType         INTEGER,
        unitLogPortOpStatus     INTEGER,
        unitLogPortAdminStatus  INTEGER
    }


    unitLogPortId OBJECT-TYPE
        SYNTAX      Integer32
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
                "The port number."
        ::= { unitLogPortEntry 1 }
```

```
    unitLogPortType OBJECT-TYPE
        SYNTAX      INTEGER {
                        tcp-logPort(1), other-logPort(2)
                    }

        MAX-ACCESS read-only
        STATUS      current
        DESCRIPTION
                "An integer that identifies the type of port."
        ::= { unitLogPortEntry 2 }



    unitLogPortOpStatus   OBJECT-TYPE
        SYNTAX      INTEGER {
                        unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                        down(5), failed(6)
                    }
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "A integer which identifies the operational status of
              the port."
        ::= { unitLogPortEntry 3  }

    unitLogPortAdminStatus   OBJECT-TYPE
        SYNTAX      INTEGER {
                        locked(1), unlocked(2)
                    }
        MAX-ACCESS      read-write
        STATUS      current
        DESCRIPTION
             "A integer which identifies the administrative status of
              the port."
        ::= { unitLogPortEntry 4  }


-- End unitLogPortTable

    unitPhysPortTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF UnitPhysPortEntry
        MAX-ACCESS not-accessible
        STATUS      current
        DESCRIPTION
                "The physical port table is a list of the physical
                 ports of a particular piece of equipment that is part
                 of a particular unit and part of a particular
                 transmission resource.

                 A corresponding row in unitEquipTable will exist
                 for rows in this table."
```

```
    ::= { unitNet 4 }

unitPhysPortEntry OBJECT-TYPE
    SYNTAX      UnitPhysPortEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "A set of parameters to identify and describe
             a physical port."
    INDEX       { transResourceId, unitId, unitEquipIndex,
                  unitPhysPortId }
    ::= { unitPhysPortTable 1  }

UnitPhysPortEntry ::= SEQUENCE {
    unitPhysPortId          Integer32,
    unitPhysPortType        INTEGER,
    unitPhysPortOpStatus    INTEGER,
    unitPhysPortAdminStatus INTEGER
}


unitPhysPortId OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "The port number.  For instance, in the case of
             an ethernet interface, this would be the index
             of the interface in the ifTable. "
    ::= { unitPhysPortEntry 1 }

unitPhysPortType OBJECT-TYPE
    SYNTAX      INTEGER {
                    ethernet-port(1), ds3-port(2), t1-port(3),
                    satcom-port(4), fddi-port(5), hssi-port(6),
                    atm-port(7)
                }
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "An integer that identifies the type of port."
    ::= { unitPhysPortEntry 2 }



unitPhysPortOpStatus   OBJECT-TYPE
    SYNTAX      INTEGER {
                     unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                     down(5), failed(6)
                }
    MAX-ACCESS      read-only
```

```
        STATUS      current
        DESCRIPTION
             "A integer which identifies the operational status of
              the port."
        ::= { unitPhysPortEntry 3  }

    unitPhysPortAdminStatus   OBJECT-TYPE
        SYNTAX      INTEGER {
                         locked(1), unlocked(2)
                    }
        MAX-ACCESS      read-write
        STATUS      current
        DESCRIPTION
             "A integer which identifies the administrative status of
              the port."
        ::= { unitPhysPortEntry 4  }

-- End unitPhysPortTable

    unitLinkTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF UnitLinkEntry
        MAX-ACCESS not-accessible
        STATUS      current
        DESCRIPTION
                "The link table is a list of the links
                 between physical ports, typically where the
                 physical ports are on different units.

                 Corresponding entries will exist in the
                 unitPhysPortTable.

                 If either end of the link resides on the local
                 unit, a corresponding row in the unitPhysPortTable
                 will be present.

                 By convention, if one end of a link is on the local
                 unit, it will be first in the unitLinkTable indexing.
                 If neither end of the link is on the local unit,
                 the unit with the lower unitId will be first in
                 the indexing."

        ::= { unitNet 5 }

    unitLinkEntry OBJECT-TYPE
        SYNTAX      UnitLinkEntry
        MAX-ACCESS not-accessible
        STATUS      current
        DESCRIPTION
                "A set of parameters to identify and describe
                 a link between physical ports."
        INDEX       { transResourceId, unitIdA, unitEquipIndexA,
                      unitPhysPortIdA, unitIdB,
```

```
                    unitEquipIndexB, unitPhysPortIdB }
    ::= { unitLinkTable 1  }

UnitLinkEntry ::= SEQUENCE {
    unitIdA              Integer32,
    unitEquipIndexA      Integer32,
    unitPhysPortIdA      Integer32,
    unitIdB              Integer32,
    unitEquipIndexB      Integer32,
    unitPhysPortIdB      Integer32,
    unitLinkId           Integer32,
    unitLinkType         INTEGER,
    unitLinkDescription  DisplayString,
    unitLinkMediaType    INTEGER,
    unitLinkSecKeyId     Integer32,
    unitLinkDistance     Integer32,
    unitLinkPower        Integer32,
    unitLinkOpStatus     INTEGER,
    unitLinkAdminStatus  INTEGER
}

unitIdA OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS     read-only
    STATUS     current
    DESCRIPTION
            "UnitId of unit on one side of link."
    ::= { unitLinkEntry 1 }

unitEquipIndexA OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS     read-only
    STATUS     current
    DESCRIPTION
            "UnitEquipIndex for the equipment on one side of link."
    ::= { unitLinkEntry 2 }

unitPhysPortIdA OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS     read-only
    STATUS     current
    DESCRIPTION
            "UnitPhysPortId of port on one side of link."
    ::= { unitLinkEntry 3 }


unitIdB OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS     read-only
    STATUS     current
    DESCRIPTION
            "UnitId of unit on other side of link."
```

```
    ::= { unitLinkEntry 4 }

unitEquipIndexB OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "UnitEquipIndex for the equipment on other side of link."
    ::= { unitLinkEntry 5 }

unitPhysPortIdB OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "UnitPhysPortId of port on other side of link."
    ::= { unitLinkEntry 6 }

unitLinkId OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "An integer indentifing the link."
    ::= { unitLinkEntry 7 }

unitLinkType OBJECT-TYPE
    SYNTAX      INTEGER {
                   csLoop(1), csTrunc(2), psLoop(3), psTrunk(4),
                   eplrs(5), jtids(6), lan(7), hfLink(8), shfLos(9),
                   shfSatcom(10), uhfLos(11), vhfLink(12)
                }

    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "An integer that identifies the type of port."
    ::= { unitLinkEntry 8 }

unitLinkDescription OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..100))
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "A description of the link."
    ::= { unitLinkEntry 9 }

unitLinkMediaType OBJECT-TYPE
    SYNTAX      INTEGER  {
                satellite(1), troposcatter(2), groundWaveLos(3),
                coaxialCable(4), multiModeFiber(5), singleModeFiber(6),
                twoWireCable(7), fourWireCable(8), thinWireEthernet(9),
```

```
                waveGuide(10), ionosphericScatterHf(11)
            }
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "An integer indentifing the link media type."
    ::= { unitLinkEntry 10 }

unitLinkSecKeyId OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "An integer indentifing security key."
    ::= { unitLinkEntry 11 }

unitLinkDistance OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "Link distance in meters."
    ::= { unitLinkEntry 12 }

unitLinkPower OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
            "Link power in watts."
    ::= { unitLinkEntry 13 }


unitLinkOpStatus   OBJECT-TYPE
    SYNTAX      INTEGER {
                    unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                    down(5), failed(6)
                }
    MAX-ACCESS      read-only
    STATUS      current
    DESCRIPTION
         "A integer which identifies the operational status of
          the link."
    ::= { unitLinkEntry 14 }

unitLinkAdminStatus   OBJECT-TYPE
    SYNTAX      INTEGER {
                    locked(1), unlocked(2)
                }
    MAX-ACCESS      read-write
    STATUS      current
    DESCRIPTION
```

```
            "A integer which identifies the administrative status of
             the link."
        ::= { unitLinkEntry 15 }

-- End unitLinkTable

    unitChanTable OBJECT-TYPE
        SYNTAX     SEQUENCE OF UnitChanEntry
        MAX-ACCESS not-accessible
        STATUS     current
        DESCRIPTION
                "The channel table is a list of the channels
                 in a link.

                 A corresponding row in unitLinkTable will exist
                 for rows in this table."

        ::= { unitNet 6 }

    unitChanEntry OBJECT-TYPE
        SYNTAX     UnitChanEntry
        MAX-ACCESS not-accessible
        STATUS     current
        DESCRIPTION
                "A set of parameters to identify and describe
                 a channel."
        INDEX      { transResourceId, unitIdA, unitEquipIndexA,
                     unitPhysPortIdA, unitIdB,
                     unitEquipIndexB, unitPhysPortIdB, unitChanId }
        ::= { unitChanTable 1  }

    UnitChanEntry ::= SEQUENCE {
        unitChanId           Integer32,
        unitChanType         Integer32,
        unitChanBandWidth    Integer32,
        unitChanPower        Integer32,
        unitChanOpStatus     INTEGER,
        unitChanAdminStatus  INTEGER
    }

    unitChanId OBJECT-TYPE
        SYNTAX     Integer32
        MAX-ACCESS     read-only
        STATUS     current
        DESCRIPTION
                "An integer indentifing the channel."
        ::= { unitChanEntry 1 }

    unitChanType OBJECT-TYPE
        SYNTAX     Integer32
        MAX-ACCESS read-only
        STATUS     current
```

```
        DESCRIPTION
                "An integer that identifies the type of channel."
        ::= { unitChanEntry 2 }

    unitChanBandWidth OBJECT-TYPE
        SYNTAX      Integer32
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
                "Channel bandwidth in kHz."
        ::= { unitChanEntry 3 }

    unitChanPower OBJECT-TYPE
        SYNTAX      Integer32
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
                "Channel power in watts."
        ::= { unitChanEntry 4 }

    unitChanOpStatus    OBJECT-TYPE
        SYNTAX      INTEGER {
                        unknown(0), c-1(1), c-2(2), c-3(3), c-4(4),
                        down(5), failed(6)
                    }
        MAX-ACCESS      read-only
        STATUS      current
        DESCRIPTION
             "A integer which identifies the operational status of
              the channel."
        ::= { unitChanEntry 5 }

    unitChanAdminStatus    OBJECT-TYPE
        SYNTAX      INTEGER {
                        locked(1), unlocked(2)
                    }
        MAX-ACCESS      read-write
        STATUS      current
        DESCRIPTION
             "A integer which identifies the administrative status of
              the channel."
        ::= { unitChanEntry 6 }

-- End unitChanTable


-- The inmNotification Group

    inmAlarmNextIndex OBJECT-TYPE
        SYNTAX      INTEGER (0..65535)
        MAX-ACCESS read-only
        STATUS      current
```

```
    DESCRIPTION
            "The index number of the next appropriate
            unassigned entry in the inmAlarmTable.  The value
            0 indicates that no unassigned entries are
            available.

            A management station should create new entries in
            the inmAlarmTable using this algorithm: first,
            issue a management protocol retrieval operation to
            determine the value of inmAlarmNextIndex; and,
            second, issue a management protocol set operation
            to create an instance of the inmAlarmStatus
            object setting its value to `createAndGo' or
            `createAndWait' (as specified in the description
            of the RowStatus textual convention)."
    ::= { inmNotification 1 }


inmAlarmTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF InmAlarmEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
            "A list of inmAlarm entries."
    ::= { inmNotification 2 }

inmAlarmEntry OBJECT-TYPE
    SYNTAX      InmAlarmEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
            "A list of parameters that determine the alarm
            condition for opStatus variables. "
    INDEX       { inmAlarmIndex }
    ::= { inmAlarmTable 1 }

InmAlarmEntry ::= SEQUENCE {
    inmAlarmIndex                   INTEGER,
    inmAlarmVariable                InstancePointer,
    inmAlarmValue                   Integer32,
    inmAlarmRisingThreshold         Integer32,
    inmAlarmFallingThreshold        Integer32,
    inmAlarmRisingEventIndex        INTEGER,
    inmAlarmFallingEventIndex       INTEGER,
    inmAlarmStatus                  RowStatus
}

inmAlarmIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..65535)
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
```

```
            "An index that uniquely identifies an entry in the
            inmAlarm table. "
    ::= { inmAlarmEntry 1 }


inmAlarmVariable OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
           "A pointer to a specific OpStatus instance."

    ::= { inmAlarmEntry 2 }


inmAlarmValue OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The value of the opStatus variable during the last
            sampling period.  This value is updated when ever
            the opStatus variable changes.

    ::= { inmAlarmEntry 3 }


 inmAlarmRisingThreshold OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
            "A threshold for the opStatus variable.  When the
            current value is greater than or equal to
            this threshold, and the value at the last sampling
            interval was less than this threshold, a single
            event will be generated.  A single event will also
            be generated if the opStatus variable changes from
            a value above this threshold to a value greater yet.


            An attempt to modify this object will fail with an
            `inconsistentValue' error if the associated
            inmAlarmStatus object would be equal to `active'
            both before and after the modification attempt."

    ::= { inmAlarmEntry 4 }


inmAlarmFallingThreshold OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-create
```

```
    STATUS      current
    DESCRIPTION
            "A threshold for the opStatus variable.  When the
            current sampled value is less than or equal to
            this threshold, and the value at the last sampling
            interval was greater than this threshold, a single
            event will be generated.

            An attempt to modify this object will fail with an
            `inconsistentValue' error if the associated
            inmAlarmStatus object would be equal to `active'
            both before and after the modification attempt."
    ::= { inmAlarmEntry 5 }


inmAlarmRisingEventIndex OBJECT-TYPE
    SYNTAX      INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS      current
    DESCRIPTION
            "The index of the inmEventEntry that is used when
            a rising threshold is crossed.  The inmEventEntry
            identified by a particular value of this index is
            the same as identified by the same value of the
            inmEventIndex object.  If there is no
            corresponding entry in the inmEventTable, then no
            association exists.  In particular, if this value
            is zero, no associated event will be generated, as
            zero is not a valid inmEventIndex.

            An attempt to modify this object will fail with an
            `inconsistentValue' error if the associated
            inmAlarmStatus object would be equal to `active'
            both before and after the modification attempt."
    ::= { inmAlarmEntry 6 }


inmAlarmFallingEventIndex OBJECT-TYPE
    SYNTAX      INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS      current
    DESCRIPTION
            "The index of the inmEventEntry that is used when
            a falling threshold is crossed.  The
            inmEventEntry identified by a particular value of
            this index is the same as identified by the same
            value of the inmEventIndex object.  If there is
            no corresponding entry in the inmEventTable, then
            no association exists.  In particular, if this
            value is zero, no associated event will be
            generated, as zero is not a valid inmEventIndex.
```

```
            An attempt to modify this object will fail with an
            `inconsistentValue' error if the associated
            inmAlarmStatus object would be equal to `active'
            both before and after the modification attempt."
    ::= { inmAlarmEntry 7 }


inmAlarmStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS read-create
    STATUS      current
    DESCRIPTION
            "The status of this inmAlarm entry.  This object
            may not be set to `active' unless the following
            columnar objects exist in this row:
            inmAlarmVariable,
            inmAlarmRisingThreshold,
            inmAlarmFallingThreshold,
            inmAlarmRisingEventIndex,
            inmAlarmFallingEventIndex"
    ::= { inmAlarmEntry 8 }


-- The inmEvent table defines the set of events generated by
-- the inm.  Each entry in the inmEventTable associates an event
-- type with the notification method and associated parameters.

inmEventNextIndex OBJECT-TYPE
    SYNTAX      INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "The index number of the next appropriate
            unassigned entry in the inmEventTable.  The value
            0 indicates that no unassigned entries are
            available.

            A management station should create new entries in
            the inmEventTable using this algorithm: first,
            issue a management protocol retrieval operation to
            determine the value of inmEventNextIndex; and,
            second, issue a management protocol set operation
            to create an instance of the inmEventStatus
            object setting its value to `createAndWait' or
            'createAndGo'."
    ::= { inmNotification 3 }


inmEventTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF InmEventEntry
    MAX-ACCESS not-accessible
    STATUS      current
```

```
    DESCRIPTION
            "A list of events."
    ::= { inmNotification 4 }

inmEventEntry OBJECT-TYPE
    SYNTAX      InmEventEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
            "A set of parameters that describe an event that
            is generated when certain conditions are met."
    INDEX       { inmEventIndex }
    ::= { inmEventTable 1 }

InmEventEntry ::= SEQUENCE {
    inmEventIndex           INTEGER,
    inmEventID              OBJECT IDENTIFIER,
    inmEventDescription     DisplayString,
    inmEventEvents          Counter32,
    inmEventLastTimeSent    TimeStamp,
    inmEventStatus          RowStatus
}

inmEventIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..65535)
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "An index that uniquely identifies an entry in the
            inmEvent table.  Each such entry defines an event
            generated when the appropriate conditions occur."
    ::= { inmEventEntry 1 }


inmEventID OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS      current
    DESCRIPTION
            "The authoritative identification of the event
            type generated by this entry.  This variable
            occurs as the second VAR bind of an InformRequest-
            PDU.  If this OBJECT IDENTIFIER maps to a
            NOTIFICATION-TYPE the sender will place the
            objects listed in the NOTIFICATION-TYPE in the
            VAR bind list."
    ::= { inmEventEntry 2 }

inmEventDescription OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..127))
    MAX-ACCESS read-create
    STATUS      current
```

```
    DESCRIPTION
            "A comment describing this inmEvent entry."
    ::= { inmEventEntry 3 }

inmEventEvents OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "The number of events caused by event generators
            associated with this inmEvent entry."
    ::= { inmEventEntry 4 }


inmEventLastTimeSent OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "The value of sysUpTime at the time this inmEvent
            entry last generated an event.  If this entry has
            not generated any events, this value will be
            zero."
    DEFVAL { 0 }
    ::= { inmEventEntry 5 }

inmEventStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS read-create
    STATUS      current
    DESCRIPTION
            "The status of this inmEvent entry.  This object
            may not be set to `active´ unless the following
            columnar objects exist in this row: inmEventID,
            inmEventDescription, inmEventEvents, and
            inmEventLastTimeSent.

            Setting an instance of this object to the value
            ´destroy´ has the effect of invalidating any/all
            entries in the inmEventTable, and the
            inmEventNotifyTable which reference the
            corresponding inmEventEntry."
    ::= { inmEventEntry 6 }


-- The inmEventNotifyTable is used to configure the
-- destination of notifications sent
-- when a particular event is triggered.

inmEventNotifyTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF InmEventNotifyEntry
```

```
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "A list of protocol configuration entries for
            event notifications from this entity."
    ::= { inmNotification 5 }

inmEventNotifyEntry OBJECT-TYPE
    SYNTAX     InmEventNotifyEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "A set of parameters that describe the type and
            destination of InformRequest-PDUs sent for a
            particular event.  The inmEventIndex in this
            entry's INDEX clause identifies the inmEventEntry
            which, when triggered, will generate a
            notification as configured in this entry.  The
            contextIdentity in this entry's INDEX clause
            identifies the context to which a notification
            will be sent."
    INDEX      { inmEventIndex, inmSessionHandleIndex }
    ::= { inmEventNotifyTable 1 }

InmEventNotifyEntry ::= SEQUENCE {
    inmSessionHandleIndex                  Integer32,
    inmEventNotifyStatus                   RowStatus
}


inmSessionHandleIndex OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "The index number of the scotty snmp handle to use
             for this notification.  This is a temporary object
             that will soon be replace by something less
             cryptic, but it is needed now to make things work."

    ::= { inmEventNotifyEntry 1 }

inmEventNotifyStatus OBJECT-TYPE
    SYNTAX     RowStatus
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
            "The state of this inmEventNotifyEntry.  This
            object may not be set to `active' unless the
            following columnar objects exist in this row:
            inmSessionHandleIndex."
    ::= { inmEventNotifyEntry 2 }
```

```
inmAlarmNotifications OBJECT IDENTIFIER ::= {inmNotification 6 }

inmRisingAlarm NOTIFICATION-TYPE
    OBJECTS { inmAlarmVariable, inmAlarmValue,
                        inmAlarmRisingThreshold }
    STATUS  current
    DESCRIPTION
            "An event that is generated when an alarm entry
            crosses its rising threshold or rises again after having
            crossed the rising threshold.  The instances of
            those objects contained within the VAR bind list
            are those of the alarm entry which generated this
            event."
    ::= { inmAlarmNotifications 1 }

inmFallingAlarm NOTIFICATION-TYPE
    OBJECTS { inmAlarmVariable, inmAlarmValue,
                        inmAlarmFallingThreshold }
    STATUS  current
    DESCRIPTION
            "An event that is generated when an alarm entry
            crosses its falling threshold The instances of
            those objects contained within the VAR bind list
            are those of the alarm entry which generated this
            event."
    ::= { inmAlarmNotifications 2 }

inmOpStatusDependencyNextIndex OBJECT-TYPE
    SYNTAX     INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The index number of the next appropriate
            unassigned entry in the inmOpStatusDependencyTable.
            The value 0 indicates that no unassigned entries are
            available.

            A management station should create new entries in
            the inmOpStatusDependencyTable using this algorithm:
            first, issue a management protocol retrieval operation
            to determine the value of inmOpStatusDependencyNextIndex;
            and, second, issue a management protocol set operation
            to create an instance of the inmOpStatusDependencyStatus
            object setting its value to `createAndGo´ or
            `createAndWait´ (as specified in the description
            of the RowStatus textual convention)."
    ::= { inmNotification 7 }

inmOpStatusDependencyTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF InmOpStatusDependencyEntry
```

```
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "A list a list of opStatus dependency entries for
            this inm."
    ::= { inmNotification 8 }

inmOpStatusDependencyEntry OBJECT-TYPE
    SYNTAX     InmOpStatusDependencyEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "Parameters that describe the  dependency between
             OpStatus variables."
    INDEX      { inmOpStatusDependencyIndex }
    ::= { inmOpStatusDependencyTable 1 }

InmOpStatusDependencyEntry ::= SEQUENCE {
    inmOpStatusDependencyIndex     INTEGER,
    dependentOpStatus              OBJECT IDENTIFIER,
    independentOpStatus            OBJECT IDENTIFIER,
    opStatusWeight                 INTEGER,
    inmOpStatusDependencyStatus    RowStatus
}

inmOpStatusDependencyIndex OBJECT-TYPE
    SYNTAX     INTEGER (1..65535)
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "An index that uniquely identifies an entry in the
            inmOpStatusDependency table. "
    ::= { inmOpStatusDependencyEntry 1 }

dependentOpStatus OBJECT-TYPE
    SYNTAX     OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "The dependent opStatus variable."
    ::= { inmOpStatusDependencyEntry 2 }

independentOpStatus OBJECT-TYPE
    SYNTAX     OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS     current
    DESCRIPTION
           "An independent opStatus variable."
    ::= { inmOpStatusDependencyEntry 3 }

opStatusWeight OBJECT-TYPE
    SYNTAX     INTEGER (1..100)
```

```
        MAX-ACCESS read-create
        STATUS     current
        DESCRIPTION
                "The weight given to this independent opStatus
                 variable.  This number should be between 1 and
                 100.  For a given dependent variable, if the weights
                 of all independent variables are less than 100, then
                 a linear weighted average is used.  If the weight
                 of at least one independent variable is 100, then the
                 dependent opStatus is calculated as the average of
                 all independent opStatus variables with weight equal
                 to 100."
        ::= { inmOpStatusDependencyEntry 4 }

    inmOpStatusDependencyStatus OBJECT-TYPE
        SYNTAX     RowStatus
        MAX-ACCESS read-create
        STATUS     current
        DESCRIPTION
                "The state of this inmOpStatusDependencyEntry.  This
                object may not be set to `active' unless the
                following columnar objects exist in this row:
                dependentOpStatus, independentOpStatus, opStatusWieght."

        ::= { inmOpStatusDependencyEntry 5 }


END
```

APPENDIX B: AICS MANAGEMENT APPLICATION CODE

The code presented in this appendix represents only a fraction of the code which makes up the entire AMA program. This code was selected and edited to give the experienced programmer a general idea of how the the AMA program works. This code is discussed in section 2 of this document.

Appendix B.1: Main Program

```
#!/usr/local/bin/scwish
###############################################################################
##
## AICS Management Application Program (Version 0.30)
##
## This program is written in the Scotty Wish  extended Tcl/Tk
## shell.  Yeah for the University of Braunschweig and all the
## people that worked on the Scotty program.
##
## This file is the main program that starts up the
## AICS Manager Application Program.  This file starts up both the
## agent and the GUI.
##
## Everett W. Jacobs May 1996
##
###############################################################################

##
## no window required for the agent
##

wm withdraw .

##
## Expand the auto_path to auto load script files when needed.
##


cd $env(AMALIB)

set auto_path [concat $scotty_library/agents $auto_path]
if {[file exists ./tclIndex]} {
    set auto_path [concat . $auto_path]
}


##
## Set some variables
##
```

```
source ./ama.conf


##
## check if invocation was correct
##

proc usage {} {
    puts stdout {usage: ama-aa [commPlan] [-interp interpreter]}
    exit
}

set commPlan $defaultCommPlan
global tkinedInterp
set tkinedInterp ""

for {set i 0} {$i < $argc} {incr i} {
    switch -exact -- [lindex $argv $i] {
        -interp {
            incr i
            if {$i < $argc} {
              set tkinedInterp [lindex $argv $i]
            } else {
              puts stderr "Missing argument following -interp switch."
              usage
            }
        }
        default {
          set commPlan [lindex $argv $i]
        }
    }
}

##
## Load the aics MIB definition, initialize AA and AMA.
##

foreach mibfile $MIBFILES {
 mib load $mibfile
}

##
## fire up an agent and initilize
##

AA_Init $commPlan

##
## Run necessary configuration procedures
##
```

```
Config_Check $configCheckList

##
## fire up the ama GUI and initilize
##

puts "Starting AMA GUI in 10 seconds..."

after 10000 exec $TOPDIR/ama_init &
```

Appendix B.2: AMA Configuration File

```
##########################################################################
#
# Configuration file for the AICS Manager Application Program
#
##########################################################################
#
# ama.conf
#
##########################################################################

#  initialize some variables

set commPlanDb_filename commPlanDb
set defaultCommPlan DEFAULT
set TOPDIR $env(AMALIB)
set MIBDIR ${TOPDIR}/../mibs
set TRAPPORT 1162
set TRAPCOMMUNITY(mlm) inmMlm
set INFORMPORT 1163
set AGENTPORT 1701

#  turn on (or off) authentication on ama to ama exchanges

set AUTHENTICATION on
##set AUTHENTICATION off

#  ordered list of mib files to load

set MIBFILES [list $MIBDIR/nrad_v2.mib $MIBDIR/aics_v2.mib \
                   $MIBDIR/aics_traps.mib  ]
## set MIBFILES [list $MIBDIR/aics_v1.mib  ]

#  ordered list of configurable items

set configCheckList {OpStatusDependency InmSnmpSession Mlm Rmon2Sim}
##set configCheckList {OpStatusDependency InmSnmpSession Mlm}
##set configCheckList {OpStatusDependency InmSnmpSession Rmon2Sim}
##set configCheckList {InmSnmpSession}
##set configCheckList {}
```

Appendix B.3: Procedure Config_Check

```
##############################################################################
#  AICS Manager Application Program
#
#  Config_Check is called from ama_aa.  Config_Check starts up the
#  configuration procedures of the configurable items.
#
##############################################################################

proc Config_Check {configCheckList} {
  global haltProcedures
  global TOPDIR

  foreach configItem $configCheckList {
    switch $configItem {

      OpStatusDependency -
      InmSnmpSession {
        Configuration $configItem
      }

      Mlm {
        global unitInmModuleType
        foreach objInst [array names unitInmModuleType] {
          if {$unitInmModuleType($objInst) == "mlm"} {
            exec $TOPDIR/mlm_mdp_script &
            lappend haltProcedures {Mlm_Halt}
            break
          }
        }
      }

      Rmon2Sim {
        global unitEquipType unitEquipIpAddress
        foreach objInst [array names unitEquipType] {
          if {$unitEquipType($objInst) == "rmon2-sim"} {
            exec $TOPDIR/rmon2sim_script $unitEquipIpAddress($objInst) &
            lappend haltProcedures {Rmon2Sim_Halt}
            break
          }
        }
      }

      default {
        puts stdout "Config_Check: Unrecognized config item: $configItem"
      }

    }
  }
}
```

Appendix B.4: Procedure Configuration

```
##############################################################################
#  AICS Manager Application Program
#
# Configuration is called from Config_Check or the scripts that are
# invoked from Config_Check.  This procedures builds the
# memberTableByPropertyGroup, propertyTableByPropertyGroup, and
# memberTableByProperty tables, builds the policy statements, and
# invokes the MDP for a given configurable item.
#
##############################################################################

proc Configuration {configItem} {

# By looking at the mib-cache the  Get_Property_Groups_Members_[...]  procedure
# creates the memberTableByPropertyGroup(propertyGroup)  array.
# The value of memberTableByPropertyGroup(propertyGroup) is a list
# of lists., e.g., if one wanted to include the hostname and port for
# a member...
#
# memberTableByPropertyGroup(hpUnix-Host) {{erebus 161} {styx 161}}
#
    Get_Property_Group_Members_[set configItem] memberTableByPropertyGroup

#
# the association between  properties and
# propertyGroup would be read in from the policy database, i.e.,
# the procedure Get_Properties reads the policy groups section of the
# policy data base to create the propertyTableByPropertyGroup(properties)
# array
#
    Get_Properties $configItem propertyTableByPropertyGroup

#
# Create the memberTableByProperty(properties) array
#
    if {[info exists memberTableByProperty]} {
      unset memberTableByProperty
    }

    foreach i [array names propertyTableByPropertyGroup] {
      foreach j $propertyTableByPropertyGroup($i) {
        if {[info exists memberTableByPropertyGroup($i)]} {
          foreach k $memberTableByPropertyGroup($i) {
            lappend memberTableByProperty($j) $k
          }
        }
      }
    }
```

```
#
# The rules for the configItem (which will be passed to the front end
# of the Mdp) are read in from the policy rules data base.
#
    set ruleList [Get_Rules $configItem]

#
# The rules are then filled in with the members (extracted from the MIB Cache
# for the current mission above) that have the appropriate properties.
#
    set policyList [Expand_Rules $ruleList memberTableByProperty]

#
# Call the Management Doctorine Program routine for the appropriate
# configItem (Mdp_$configItem).  Mdp_$configItem will interprete all the
# policies in the policyList and proceed to perform
# the required configuration.
#
    Mdp_[set configItem] $policyList

}
```

Appendix B.5: Properties Data Base File

```
#########################################################################
# Properties data base file
#
# Lines in this file starting with "#" or "%" are comments.
# Blank lines are ignored.
#
# The first string in each line of this file is a property group.
# All additional strings on each line are properties in the property group.
# In this case, each property group only has one property.  In general,
# each property group can have many properties.
#
# #######################################################################
#
# inmsnmpsessionProperties
#
# #######################################################################
#
sup_primary-roc          sup_primary-roc-snmpSession
sup_secondary-roc        sup_secondary-roc-snmpSession
sup_other-roc            sup_other-roc-snmpSession
sup_primary-noc          sup_primary-noc-snmpSession
sup_secondary-noc        sup_secondary-noc-snmpSession
sup_other-noc            sup_other-noc-snmpSession
sub_primary-loc          sub_primary-loc-snmpSession
sub_secondary-loc        sub_secondary-loc-snmpSession
sub_other-loc            sub_other-loc-snmpSession
sub_primary-roc          sub_primary-roc-snmpSession
sub_secondary-roc        sub_secondary-roc-snmpSession
sub_other-roc            sub_other-roc-snmpSession
sub_primary-noc          sub_primary-noc-snmpSession
sub_secondary-noc        sub_secondary-noc-snmpSession
sub_other-noc            sub_other-noc-snmpSession
```

Appendix B.6: Policy Rules File

```
##############################################################################
# Policy Rule File
#
# Lines in this file starting with "#" or "%" are comments
#
# the syntax of a line in this file is...
#
# Alabels:propertyA [Blabels:propertyB ...] {policy_template}
#
# where Alabels has the form
#    labelA1[,labelA2...]
#
# which means that "policy_template" will be replicated once for each
# "member" (usually a network addressable unit) which is associated with
# a property group which includes propertyA (or propertyB).  The member
# will be substituted for each occurrence of the label in the template.
# The label must be separated from other alphanumeric characters in the
# template by at least one non-alphanumeric character.
#
#
##############################################################################
#
# inmsnmpsessionPolicyRules
#
##############################################################################
#
# Policies for setting up INM SNMP sessions:
#
# r is the ip address
# p is the port
# i is the table index
#
# policy templates:
#
# {session_config r p i <context> <manager|agent>}
# {inform_config i <opStatus object> <rising threshold> <falling threshold>}
# {poll_config i  <opStatus object> <polling interval (seconds)>}
#
r,p,i:sup_primary-roc-snmpSession {session_config r p i max-access agent}
r,p,i:sup_primary-roc-snmpSession {inform_config i missionOpStatus 3 1 }
r,p,i:sup_primary-roc-snmpSession {inform_config i userAppOpStatus 3 1 }
r,p,i:sup_primary-roc-snmpSession {inform_config i commServiceOpStatus 3 1 }
r,p,i:sup_primary-roc-snmpSession {inform_config i transResourceOpStatus 3 1 }
r,p,i:sup_secondary-roc-snmpSession {session_config r p i readOnly agent}
r,p,i:sup_other-roc-snmpSession {session_config r p i noSec-readOnly agent}
r,p,i:sup_primary-noc-snmpSession {session_config r p i max-access agent}
r,p,i:sup_primary-noc-snmpSession {inform_config i missionOpStatus 3 1 }
r,p,i:sup_primary-noc-snmpSession {inform_config i userAppOpStatus 4 1 }
```

```
r,p,i:sup_primary-noc-snmpSession {inform_config i subOpStatus 3 1 }
r,p,i:sup_secondary-noc-snmpSession {session_config r p i readOnly agent}
r,p,i:sup_other-noc-snmpSession {session_config r p i noSec-readOnly agent}
r,p,i:sub_primary-loc-snmpSession {session_config r p i max-access manager}
r,p,i:sub_primary-loc-snmpSession {poll_config i subOpStatus 60}
r,p,i:sub_secondary-loc-snmpSession {session_config r p i readOnly manager}
r,p,i:sub_other-loc-snmpSession {session_config r p i noSec-readOnly manager}
r,p,i:sub_primary-roc-snmpSession {session_config r p i max-access manager}
r,p,i:sub_primary-roc-snmpSession {poll_config i subOpStatus 60}
r,p,i:sub_secondary-roc-snmpSession {session_config r p i readOnly manager}
r,p,i:sub_other-roc-snmpSession {session_config r p i noSec-readOnly manager}
r,p,i:sub_primary-noc-snmpSession {session_config r p i max-access manager}
r,p,i:sub_secondary-noc-snmpSession {session_config r p i readOnly manager}
r,p,i:sub_other-noc-snmpSession {session_config r p i noSec-readOnly manager}
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1997	Final: Dec 1995 – Sep 1996

4. TITLE AND SUBTITLE

AUTOMATED INTEGRATED COMMUNICATIONS SYSTEMS (AICS) INTEGRATED NETWORK MANAGER PROTOTYPE DOCUMENTATION

5. FUNDING NUMBERS

PE: 0603794N

6. AUTHOR(S)

E. W. Jacobs, M. E. Inchiosa, L. M. Gutman, C. T. Barber

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division
San Diego, California 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER

TR 1735

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Space and Naval Warfare Systems Command
SPAWAR PD 13
2451 Crystal Drive
Arlington, VA 22245–5200

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report documents the Integrated Network Manager (INM) prototype development and integration effort, undertaken as part of the FY 96 Automated Integrated Communications Systems (AICS) program. The AICS architecture describes a hierarchy of INMs, where local operations centers (LOCs), the INMs at the lowest level of the hierarchy, are responsible for the communications/network assets under their purview. The AICS architecture is primarily targeted for an environment where INMs are commonly separated from other INMs by RF links. The INM prototype work centered on development of the AICS Management Application (AMA). The ultimate vision for AICS is a highly automated network management system where its functions are implemented using standards-based management protocols, and where many operations are carried out in a virtually unattended mode. The prototype INM provides a working example demonstrating aspects of this ultimate vision, and provides a starting point for development of a network management system that can evolve with new technology.

14. SUBJECT TERMS

Mission Area: Communications
network management
simple network management protocol (SNMP)

15. NUMBER OF PAGES

88

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
E. W. Jacobs	(619) 553-1614 jacobs@nosc.mil	Code D364

INITIAL DISTRIBUTION

Code D0012	Patent Counsel	(1)
Code D0271	Archive/Stock	(6)
Code D0274	Library	(2)
Code D0271	D. Richter	(1)
Code D364	E. W. Jacobs	(1)
Code D805	M. S. Kvigne	(1)
Code D82	R. J. Kochanski	(1)
Code D8205	K. R. Casey	(1)
Code D827	L. W. Gutman	(1)
Code D827	R. L. Merk	(1)
Code D8405	B. J. Marsh	(1)
Code D8505	R. D. Peterson	(1)

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (4)

NCCOSC Washington Liaison Office
Washington, DC 20363–5100

Center for Naval Analyses
Alexandria, VA 22302–0268

Navy Acquisition, Research and Development
Information Center (NARDIC)
Arlington, VA 22244–5114

GIDEP Operations Center
Corona, CA 91718–8000

Space and Naval Warfare Systems Command
Arlington, VA 22245–5200

Space and Naval Warfare Systems Command
San Diego, CA 92152–5002 (2)

Naval Command, Control and Ocean
Surveillance Center, In-Service
Engineering, East Coast Division
(NISE East) (3)